Occasional Publications of the
DEPARTMENT OF GEOGRAPHY
UNIVERSITY OF ILLINOIS
at Urbana-Champaign

1.00	.98	.41	.89
	1.00	.63	.76
		1.00	.64
			1.00

23.2	.71	1002	8.7
39.4	.29	996	9.2
51.3	.31	1291	3.1
•	•	•	•
•	•	•	•
•	•	•	•
28.1	.66	771	2.1

AN INTRODUCTION TO FORTRAN AND THE PROGRAMMING OF SPATIAL DATA

by

J. Ronald Eyton
Curtis C. Roseman

PAPER NUMBER 13
July, 1979

Editor of Occasional Papers: Rick Mattson

GEOGRAPHY GRADUATE STUDENT ASSOCIATION
UNIVERSITY OF ILLINOIS at Urbana-Champaign

An Introduction to FORTRAN and the PROGRAMMING OF SPATIAL DATA

by

J. RONALD EYTON
Department of Geography
University of South Carolina

and

CURTIS C. ROSEMAN
Department of Geography
University of Illinois

Occasional Publication Number 13
Department of Geography
University of Illinois
Urbana, Illinois
July, 1979

TABLE OF CONTENTS

SECTION II: FORTRAN PROGRAMMING OF SPATIAL DATA

PREFACE

This manual was written as an introduction to the basics of FORTRAN programming and as an introduction to the computer analysis of spatial data. It contains very concise explanations and therefore is intended to be used either as a reference work or as a framework and supplement for class room lectures. In the classroom this manual might be used in conjunction with statistical or quantitative methods courses in geography or other fields, or with courses in spatial analysis or spatial programming.

The eight lessons of Section I contain all that is needed to perform basic manipulations of data matricies spatial or non-spatial. The lessons also employ programming examples and exercises based on simple statistics and a summary of statistical formulae and tests are included in Appendix A for reference. Section II introduces the student to a range of computer procedures necessary for the manipulation and display of spatial data, including trend surface analysis, line printer mapping, digitizer utilization, and the programming of a line plotter.

ACKNOWLEDGEMENTS

We would like to thank the many persons who have contributed in one way or another to the completion of this document, including several students in our classes over the last six years. Special thanks goes to Howard G. Roepke who helped critique the programming sections, to Elizabeth Mercer Roseman who wrote the first draft for portions of Section I, and Jerome D. Fellmann and Marc Armstrong who provided editorial assistance.

SECTION I

FORTRAN PROGRAMMING

INTRODUCTION

Section I consists of eight basic lessons on Fortran IV, a computer language based on mathematical logic. It is used to instruct the computer how and where to read data, how to manipulate these data, and how to arrange the results of the calculations on the printout sheets.

Card Deck

The original data and programmed instructions are normally fed into the computer on computer cards (tapes and discs may also be used). There are three types of computer cards: (1) Job Control Cards, (2) Fortran Statement Cards, and (3) Data Cards. Each type conveys different information to the computer and must be arranged in a certain order.

1. Types and Purposes of Cards:
   a. Job Control Cards - instruct the computer to look at the Fortran Statement Cards, stop, look at data cards, stop, etc.
   b. Fortran Statement Cards - instruct the computer to perform a particular set of instructions.
   c. Data Cards - contain original data.
2. Order of Cards:
   a. Job Control
   b. Fortran Statements
   c. Job Control
   d. Data Deck
   e. Job Control

Fortran Statements

The programming portion of the deck consists of three parts: (1) statements to READ data; (2) statements for CALCULATIONS and MANIPULATIONS, and (3) statements to WRITE results. In the program there can only be one statement per card and this must be punched in columns 7-72. Statement numbers which are used to identify certain statements throughout the program can range in value from 1-9999 and must be punched in columns 2-5.

Statements must be arranged a certain way. The computer will execute each statement (i.e., do what the statement instructs it to do) in the order in which it is found in the deck. (Later, however, it will be shown that there are ways of instructing the computer to transfer control, i.e., "skip around," to different statements in different locations within the deck.)

The following is a list of the column uses for a Fortran statement on a standard 80-column card. Instructions in the use of all of these columns will be presented as the need arises.

Columns	
1	Comment Card Indication
2-5	Statement Numbering
6	Card Continuation Indication
7-72	Fortran Statement
73-80	Card Sequencing

LESSON 1: BASIC FORTRAN STATEMENTS

Focus:	How to read in data and write it out.
New Statements:	READ, WRITE, FORMAT, STOP, END
Exercise:	Reading and writing data

Introduction

The statements introduced in this lesson are usually used in the first and last parts of a Fortran Program. The READ statement directs the computer to read the data. The WRITE statement tells the computer what to print out. The FORMAT statement assigns the form in which the data are to be read in and printed out. Finally, the STOP and END statements tell the computer the program is done.

Basic Fortran Statements

READ Statement

```
Card COLS     1234567
Fortran             READ (5,aaaa) variable name, variable name . . .
Statements     aaaa FORMAT (     )
```

Explanation:

5 = indicates that a card will be read

a = statement number that identifies the format card for reading in the data

variable name = each variable in the original data must have its own identification name. Each name must satisfy the following convention:

1. the name must have six or fewer characters

2. the name may be alphabetic or numeric but the first character must be alphabetic

3. there are two types of names corresponding to numerical data types:

INTEGERS	I-N (whole numbers)
FLOATING POINT	A-H; O-Z (decimals)

Correct Examples:

B292	(floating point)
RATE	(floating point)
MQ73	(integer)
A	(floating point)
ID	(integer)

Incorrect Examples:

ATIKOKAN	(more than <u>6</u> characters)
65A	(starts with a number)
C$/	(can only use letters and numbers)

<u>WRITE Statement</u>

```
Card COLS     1234567
Fortran             WRITE (6,aaaa)XX,ZERO,JT,CLYDE,MARY,II,RATE
Statements     aaaa FORMAT (     )
```

Where:

a = statement number that identifies format card for writing out the data

6 = unit number for printed output (on lister sheet)

<u>FORMAT Statement</u>

FORMAT statements describe the arrangement of data on cards so the computer may correctly read them, or specify the arrangement of data output on the printed page.

Data input and output are arranged in "fields." A field is a group of columns which contain a single piece of data.

Example 1: Formats for READ Statements

1. If data are in integer form use Ia; if data are in floating point form use Fa.b.

   Where: a = size of field (includes numerals, decimal points, and plus and minus signs)

   b = number of digits to the right of the decimal

2. If skipping a space between fields use 1X; two spaces use 2X; etc.

3. Example:

```
TO READ THIS DATA CARD

Card Column 1    8    13  17      25      35
            ↓    ↓    ↓   ↓       ↓       ↓
            24.5 2.6  9   100.00  -56.896 78

USE THESE READ AND FORMAT STATEMENTS

Card Column 1234567
                  READ (5.500)XX,ZERO,JT,CLYDE,RATE,MARGE
             500  FORMAT (F4.1,3X,F3.1,2X,I1,3X,F6.2,2X,F7.3,3X,I
```

These statements will instruct the computer to read a number with the name XX and a value of 24.5, skip 3 spaces, read a number with the name ZERO and a value of 2.6, skip 2 spaces, etc.

Example 2: Formats for WRITE Statements

1. The form used in WRITE Statements is the same used in READ, except:

   a. leave at least one additional space in each field for a plus or minus sign

b. must use carriage control which tells the computer how to print on the page

"1" = start at top of new page

" " = single space

"0" = double space

"_" = triple space

"+" = overprint

2. Example:

```
Card Column    1234567
Fortran              WRITE(6,600)ZZ,ZERO,JT,CLYDE,RATE,MARGE
Statements       600 FORTRAN("0",F5.1,3X,F4.1,2X,I2,3X,F7.2,2X,
                             F8.3,3X,I3)
```

This format will instruct the computer to double space (skip one line) and print out a value for the floating point variable ZZ in a field of size 5, skip 3 spaces, print a value for the variable ZERO, etc.

NOTE: There are 132 spaces per line available for written output.

Termination Statements

Every program needs termination statements to indicate to the computer that the program is ended.

```
Card Column  1234567
Fortran            STOP
Statements         END
```

Exercise

This problem deals with reading in some data and writing it out. Read in seven numbers which are punched on one card in fields spaced in any manner on the card. Include various sized numbers--some floating point numbers and some integer numbers. Have the data written out in reverse order.

LESSON 2: READING, WRITING AND PUNCHING

Focus:	Skipping cards or lines, standardized formats, punched output and literal messages
New Statements:	None
Exercise:	Reading, writing and punching data

Introduction

This lesson discusses new format commands that are useful tools when formatting READ and WRITE statements. This allows the user to read in data from more than one card, write out results on more than one line, and punch out results on more than one card.

Further Formatting

Skipping Cards or Lines

The / is utilized in READ formats to instruct the computer to read the next data card while in WRITE formats it indicates a movement to the next line or card. The / takes the place of a comma in the format statement.

```
/   = go to next card or skip a line
//  = skip two lines
/// = skip three lines, etc.
```

Example:*

```
      READ(5,501)ZZ,ZERO,JT,CLYDE,ART,MARGE
  501 FORMAT(F4.1,3X,F3.1/2X,I1,3X,F6.2/2X,F7.3,3X,I2)
```

This will direct the computer to read the value of ZZ, skip three spaces, read ZERO, go to next card, skip two spaces, read JT, skip three spaces, read CLYDE, go to next card, etc.

```
      WRITE(6,601)ZZ,ZERO,TJ,CLYDE,ART,BARGE
  601 FORMAT("1",F8.4,2X,F8.4,2X,F8.4/1X,F8.4,2X,F8.4,2X,F8.4)
```

In this case, the computer will begin on a new page, write the value of ZZ, skip 2 spaces, write a value for ZERO, skip 2 spaces, write a value for TJ, skip a line, skip 1 space, write a value for CLYDE, etc.

Standardized Formats

If the data are punched in regularly sized and spaced fields on a card or if the results are to be all written in regular form, the formats can be aggregated.

Example:

```
      READ(5,502)ZZ,ZERO,TJ,CLYDE,ART,BARGE
  502 FORMAT(3(F7.3,3X)/3(F7.3,3X))
```

Here the computer will alternately read three fields of F7.3 and skip three spaces, go to the next card and repeat. Thus, the three preceding any commands in parentheses indicates that those commands should be done three times.

---

*In the remainder of the lessons, card columns will not be specified--but remember that Fortran Statements cannot begin before column 7 and that statement numbers are in column 2-5.

Other Examples:

```
503 FORMAT(2F3.2,5F7.3,3(I2,5X))
```

NOTE: Parentheses are not needed to repeat a single command— 2F3.2.

```
504 FORMAT(3(2X,F4.1,I3/),F3.2)
```

NOTE: A / may also be repeated.

Example:

```
    WRITE(6,602)T1,T2,T3,ART1,ART2,ART3
602 FORMAT("0",3(F8.4,2X)/1X,3(F8.4,2X))
```

NOTE: Carriage control skipped for the second line.

Punched Output

To obtain punched output, replace the 6 in the WRITE statement with a 7. For punched output there are 80 columns available per card. A carriage control is not needed in the accompanying FORMAT for punched output.

Example:

```
    WRITE(7,700)ZZ,ZERO,TJ,CLYDE,R292,BARGE
700 FORMAT(5F10.2/5I0.2)
```

Literal Messages

A message, title or header can be written on the output sheet by including the message in the WRITE format statement. The message is contained between a pair of quotes and all characters including blanks can be used to write the message.

Example: (message only)

```
      WRITE(6,101)
  101 FORMAT("1","WHILE ART IS AWAY, THE MICE WILL PLAY")
```

Exercise

READ in 10 numbers from 3 cards. Mix integer and floating point numbers. Write out a message (header) and the numbers on two rows of print-out paper. Punch the numbers out on 4 cards.

LESSON 3: MESSAGES AND CALCULATION

Focus:	Reading in literal statements and basic calculations
New Statements:	Fortran algebraic expressions for addition, subtraction, division, multiplication, exponentiation, and square root
Exercise:	Reading in a title and reading in data for calculations

Introduction

Letters and numbers used only as symbols may be read in or written out using an Alphanumeric format. This is useful for titling or labeling output. The second half of the lesson introduces the user to the Fortran equivalent of basic algebraic manipulations.

Alphanumeric Formatting (A Formats)

A formats are used when reading in words or numbers that are only used as symbols. Numbers read in under an A format cannot be manipulated algebraically. The following message: "Lesson #3 - Exercise in Simple Calculations" punched in columns 10-53 of the first data card can be read in as follows; first designate a variable name for each set of four letters and blank spaces.

Example:

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
LESS	ON_#	3_-_	EXER	CISE	_IN_	SIMP	LE_C	ALCU	LATI	ON

To format this message, utilize the A Format which is the letter A followed by a digit representing the number of letters and spaces in each set. In this case, the digit would be 4 except for the last character set which would be 2. (Allowable A Formats are A1, A2, A3, A4)[1]

Example:

```
      READ(5,507)T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11
  507 FORMAT(9X,A4,A4,A4,A4,A4,A4,A4,A4,A4,A4,A2)
```

Or, better:

```
  507 FORMAT(9X,10A4,A2)
```

To write out the above title (header) on the top of a new page use the following:

```
                              WRITE(6,508)T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11
Title left justified      508 FORMAT("1",10A4,A2)
Title centered            508 FORMAT("1",40X,10A4,A2)
```

Calculations

Calculations are carried out in the main body of the Fortran Program. Each calculation consists of 3 parts: a left-hand side which consists of a single variable name to which the result of the calculation is assigned, an equals sign and a right-hand side consisting of one or more variables and the appropriate Fortran calculation symbol. The general form is shown below followed by the Fortran calculation symbols.

Assigned Variable Name = Variable Name, Calculation Symbol, Variable Name

---

[1]This will vary with computer systems.

The Fortran calculation symbols are:

Operation	Calculation Symbol
addition	+
subtraction	-
multiplication	*
division	/
exponentiation	**
square root	SQRT (variable)

Examples:

Addition, subtraction, multiplication, division

```
SUM = A+B
ISUM = N1+N2
DIF = A-B
PROD = A*B
DIV = A/3.0
```

Exponentiation, square root

```
CUBE = A**3
SQUARE = A**2
ROOT3 = A**1./3
ROOT2 = A**.5
ROOT2 = SQRT(A)
```

Mathematical operations may be combined in many ways, using parentheses to separate various combinations. All operations are performed in a hierarchial sequence, exponentiation takes place first, then multiplication and division, then the addition and subtraction. Operations within parentheses are completed first.

Example:

```
A    = (B+C)/3.0
B1   = SQRT(B+C/A))
SUMT = (C*(A+B))**2
```

Exercise

Read in a title for this problem from a data card. Read in some data and add, subtract, divide, multiply, square, and take the square root of some of the numbers. Write out the title. Write out answers in column or table form, being sure to label them.

LESSON 4: LOOPS, COUNTERS AND SUMMING

Focus:	Assigning the computer to perform the same set of operations over and over for a number of iterations
New Statements:	CONTINUE, IF, GO TO
Exercise:	Calculate mean and standard deviation of a data set

Introduction

The calculating advantage of a computer lies in its ability to perform a large number of tasks repeatedly and quickly. The concept of repeated operations is fundamental to programming and allows for the counting and summing operations necessary for sophisticated mathematical manipulation.

New Statements

CONTINUE Statement

The CONTINUE statement is a "convenience" statement allowing the programmer to transfer control to part of the program and then continue at that point. When the program encounters a CONTINUE statement, the program will simply go on to the next statement immediately following the CONTINUE. In this sense, it is a dummy statement that is used only as a reference point.

IF Statement

IF statements check to see if a certain condition is met by elements of the data, e.g., is A1 greater than A5? This is written:

```
IF(A1.GT.A5) a
```

Where a is an executable Fortran statement and GT signifies the condition "greater than." If the condition is met, a will be executed, if not, the next Fortran statement (following the IF statement) will be executed.

Other possible conditions are:

1. .LT. = less than
2. .EQ. = equals to
3. .NE. = not equal to
4. .LE. = less than or equal to
5. .GE. = greater than or equal to

These conditions can be combined using AND or OR as follows:

```
IF(A1.EQ.A4.AND.A2.EQ.A3) A = B
IF(A1.EQ.A4.OR.A2.EQ.A3)  I = I + 1
```

NOTE: Be sure to put a period before and after every condition.

GO TO Statement

GO TO is a branching mechanism that directs the computer to a new statement located anywhere in the program.

Example:

```
   IF(A1.GT.A5) GO TO 10
   GO TO 11
10 A6 = 10.0
   GO TO 12
11 A6 = 5.0
12 CONTINUE
```

Here, if A1 is in fact greater than A5, the computer will go to statement #10 and A6 will be set equal to 10.0; if not, it will go to statement #11 and A6 will be assigned the value 5.0.

Loops

The general form of a loop is as follows:

```
           8 READ_______________________
             FORMAT_____________________
             ___________________________  (Fortran Statements)
Loop         ___________________________
             WRITE______________________
             FORMAT_____________________

             GO TO 8
          10 STOP
             END
```

Here, a series of statements is executed repeatedly. In other words, the loop tells the computer to read the first card, do certain calculations, and write out the results. Then it returns to the beginning of the program (GO TO 8), this time reading, calculating, and writing for the second card (observation). This process will continue indefinitely as there is no mechanism to exit the loop.

Incremental Counters

In the previous example of a program with a loop, the program will iterate indefinitely (or until the data cards run out). To end the loop the program must be able to keep track of the number of times the loop iterates and then exit the loop at the appropriate count. This is done by creating an incremental counter.

NOTE: In the Fortran language a variable name can be assigned a value using a simple algebraic statement.

Integer variables set to one and zero:

I = 0

J = 1

Also in Fortran a variable can be redefined in terms of itself. A series of statements such as shown below are legitimate Fortran statements.

```
1    I = 0
2    I = I + 1
3    I = I + 1
```

After the program has executed statement 1, I = 0; after executing statement 2, I = 0 + 1 and, therefore, I = 1; and after executing statement 3, I = I + 1 or I = 2. Note that in statements 2 and 3 the I on the right-hand side of the equation takes on the value of the previously defined I before assigning a value to the left-hand side I. A statement in which an integer variable name is set equal to itself and some constant value is called an incremental counter. This statement can be used to control the exit from a loop.

Example:

Write a program to read in 100 data cards with one number per card and write out these numbers.

```
                                  I = 1 <-- (COUNTER INITIALIZED OUTSIDE OF LOOP)
                    ---------> 10 READ(5,11) A1
                    |          11 FORMAT(F10.2)
                    |             WRITE(6,21)A1
            Loop    |          21 FORMAT(1X,F10.2)
                    |             IF(I.EQ.100) GO TO 30 <--(CHECK TO EXIT FROM LOOP)
                    |             I = I + 1 <--(COUNTER)
(will iterate 100   -----------   GO TO 10
   times)                      30 CONTINUE
```

This is the standard structure for creating a loop with a GO TO statement. Note the following in the example:

1. The counter (I) must be initialized (given an integer value) outside of and before the top of the loop.
2. The counter is incrementing (SUMMING) as it keeps redefining its value in terms of itself +1.
3. After the 100th card is read, I = 100 and the looping will cease as the program transfers to statement 30.
4. If the statement I = I + 1 preceded the IF statement, only 99 cards would be read.

Summing

The fact that a variable may be redefined in terms of itself in the Fortran language allows for incrementing or summing. If this procedure is used for determining the number of times a loop iterates, it is called a counter. However, the same procedure can be used to sum a group of numbers.

Example:

Using the previous program to read in and write out 100 number, also total the 100 numbers and write out the sum.

```
                SUM = 0.          <--  (SUM INITIALIZED TO ZERO)
                I = 1
      ----->10  READ(5,11) A1
     |      11  FORMAT(F10.2)
     |          WRITE(6,21) A1
Loop |      21  FORMAT(1X,F10.2)
     |          SUM = SUM + A1    <--  (SUMMING)
     |          IF(I.EQ.100) GO TO 30
     |          I = I + 1
      ----------GO TO 10
            30  CONTINUE
                WRITE(6,31) SUM   <--  (WRITE AND FORMAT
            31  FORMAT (1X,"SUM=",F10.2)STATEMENTS FOR SUM
                                        OCCURS OUTSIDE AND
                                        AFTER LOOP TERMINATED)
```

Exercise

Read in the 30 observations of weather information from the data table. Read in the observation year under an alphanumeric format.

Write out the information in a list along with the mean and standard deviation of the observations.

Grand Forks, N.D. January Average Temperatures (^{o}F.)

1931	16.2	1941	7.2	1951	0.9
1932	9.4	1942	16.6	1952	-0.2
1933	9.4	1943	-3.4	1953	9.9
1934	10.6	1944	18.0	1954	-3.2
1935	-2.2	1945	8.8	1955	6.9
1936	10.1	1946	5.6	1956	4.4
1937	-9.7	1947	12.6	1957	0.7
1938	3.7	1948	1.8	1958	15.1
1939	7.6	1949	1.4	1959	-0.9
1940	3.0	1950	-11.3	1960	7.1

LESSON 5: STORING DATA INTERNALLY

Focus:	How to store data in the computer so it can be used by the program
New Statement:	DIMENSION
Exercise:	Repeat of previous exercise with the use of internal storage

Introduction

The use of internal program storage of data allows for the separation of a program into logical steps. Separate loops for reading, calculating and writing lessens the confusion in the logical construction of a useful algorithm.

Single Subscripted Variables

Data may be read into a single subscripted variable (vector) for storage in the computer.

```
Example:   A(1)    B(1)    C(1)    D(1)
           A(2)    B(2)    C(2)    D(2)
           A(3)    B(3)    C(3)    D(3)
            .       .       .       .
            .       .       .       .
            .       .       .       .
           A(N)    B(N)    C(N)    C(N)
```

Here we have 4 vectors each of length N and each with one variable name. Individual elements of each vector are assigned a subscript.

Data stored in this form can be accessed by designating the variable name plus the integer that corresponds to the subscript [example: A(5)]

or by designating the variable name plus an integer name that has been assigned the appropriate value of the subscript [example: A(I) where I has been assigned the value of 5].

Use of the subscripted variable is shown in the following examples. First, all the data can be read and stored in the computer, then all the calculations can be done and stored, and then all the results can be written out. Thus, three loops are used. The general form of such a routine is as follows:

```
                          DIMENSION X(100)
                          READ(5,11)NOBS  <- NOBS IS THE NUMBER OF DATA
                     11   FORMAT(I3)              CARDS TO BE READ IN
                          J=1
               ┌──> 100   READ(5,111) X(J)
      READ     │    111   FORMAT(1X,F10.2)
      LOOP     │          IF(J.EQ.NOBS) GO TO 150
               │          J=J+1
               └───────── GO TO 100
                    150   CONTINUE
                          SUM=0
                          J=1
               ┌──> 200   SUM=SUM+X(J)
      SUMMING  │          IF(J.EQ.NOBS) GO TO 250
      LOOP     │          J=J+1
               └───────── GO TO 200
                    250   CONTINUE
                          J=1
               ┌──> 300   WRITE(6,111) X(J)
      WRITE    │          IF(J.EQ.NOBS) GO TO 350
      LOOP     │          J=J+1
               │          GO TO 300
               └─── 350   CONTINUE
```

DIMENSION Statements

This statement directs the computer to reserve storage space for a subscripted variable. The DIMENSION statement is usually the first statement of a program.

Example: DIMENSION A(20),B(20),C(20),SUM(20)

In this example, each of four subscripted variables is given 20 storage locations.

Exercise

Do the previous assignment over again using three loops: loop 1 for reading in and storing the data, loop 2 for calculating the sums, and loop 3 for writing out the data and the results.

LESSON 6: DO LOOPS

Focus:	How to simplify loops by using DO statements
New Statement:	DO
Exercise:	Simple linear regression and correlation program

Introduction

The use of a DO statement eliminates the need for directly establishing an incremental counter and a GO TO statement for looping. The DO statement is the "workhorse" statement of FORTRAN programming.

DO Loops

The DO statement is placed at the beginning of a loop and specifies: (1) the location of the bottom of the loop and (2) the number of times the statements within the loop are to be executed.

The simplest form of the DO loop is:

```
DO a I = 1,N
```

Where a is the statement number that identifies the bottom of the loop, I is the counter, 1 is the initial value of the counter, and N designates the number of times the loop will iterate. All statements down to and including the one numbered "a" will be executed N times. Each time through, "I" will be incremented by one. The last time through the loop "I" will have the value of N and the loop will terminate.

Example: Finding the mean of data vector B (n=20).

```
   SUMB = 0
   DO 10 I = 1,20
10 SUMB = SUMB + B(I)
   XMEANB = SUMB/20.
```

In this example, statement 10 will be executed 20 times resulting in the sum of the twenty numbers. After the loop terminates the next statement (calculating the mean) immediately following the bottom of the loop will be completed.

The above case shows a counter (I) incrementing by 1. To consider every 2nd or 3rd value, etc., of a subscripted variable the counter can be incremented by 2 or 3, etc. The full form of a DO loop is:

```
DO 10 I = a,20,b
```

Where: a = initial value of the counter

b = counter increment each time the computer goes through the loop (iterates).

Example:

```
   DO 10 I = 2,20,2
   .
   .
   .
10 CONTINUE
```

Here, the computer will first consider the second value of a subscripted variable. Then the second time it goes through the loop, it will increment by 2 and consider the 4th value and so on until the 20th value is used in the loop.

Example: Write a program to calculate the mean and standard deviation of a variable having up to 100 observations.

Formulae:

$$\text{Mean} = \frac{\Sigma X}{N}$$

$$\text{Standard Deviation} = \sqrt{\frac{\Sigma X^2}{N} - (\overline{X})^2}$$

To use both of these formulae, the ΣX and the ΣX^2 are needed. An additional requirement is to make the program general and able to handle up to 100 observations, assuming one observation per data card.

The program using DO loops is broken into three parts:

1. Reads the necessary information
2. Calculates ΣX, ΣX^2, $\overline{X}$, σ
3. Writes out data and results

```
C     PROGRAM TO CALCULATE MEAN AND STANDARD DEVIATION
      DIMENSION X(100)
C     READ IN NO. OF OBS.
      READ(5,21) NOBS
   21 FORMAT(I3)
      XNOBS=NOBS
C     READ IN DATA
      DO 30 J=1, NOBS
   30 READ(5,31) X(J)
   31 FORMAT(F10.2)
C     CALCULATE SUMX AND SUMX2
      SUMX=0.
      SUMX2=0.
      DO 40 J=1, NOBS
      SUMX=SUMX + X(J)
   40 SUMX2=SUMX2 + X(J)**2
C     CALCULATE MEAN AND STD. DEV.
      XMEAN=SUMX/XNOBS
      STDEV=SQRT(SUMX2/XNOBS-XMEAN**2)
C     WRITE OUT DATA
      DO 50 J=1, NOBS
   50 WRITE(6,51) X(J)
   51 FORMAT(1X,F10.2)
C     WRITE OUT MEAN AND STD. DEV.
      WRITE (6.61)XMEAN, STDEV
   61 FORMAT(////1X,"MEAN=",F10.2,//1X,"STDEV =",F10.2)
      STOP
      END
```

Several other programs showing the use of DO loops are found in

Appendix B. Note that "comment" cards (those with C in column 1) are used in this program to identify different parts of the program. These are nonexecutable statements included solely for the programmers future reference.

Exercise

Write a program to calculate the simple linear correlation coefficient and regression coefficients for N pairs of observations. A guide to the structuring of such a program is shown below.

<u>Regression and Correlation Formulae</u>

$$\hat{Y} = a+bX$$

$$r = \frac{N\Sigma XY - (\Sigma X)\ (\Sigma Y)}{\sqrt{[N\Sigma X^2 - (\Sigma X)^2]\ [N\Sigma Y^2 - (\Sigma Y)^2]}}$$

$$a = \frac{(\Sigma Y)\ (\Sigma X^2) - (\Sigma X)\ (\Sigma XY)}{N\Sigma X^2 - (\Sigma X^2)}$$

$$b = \frac{N\Sigma XY - (\Sigma X)\ (\Sigma Y)}{N\Sigma X^2 - (\Sigma X)^2}$$

<u>Program Steps</u>

1. Read in data X(J), Y(J)

2. From the formula above you will need:

$$\left.\begin{array}{l} \Sigma XY = \mathrm{SUMXY} \\ \Sigma X \ \ = \mathrm{SUMX} \\ \Sigma Y \ \ = \mathrm{SUMY} \\ \Sigma X^2 = \mathrm{SUMX2} \\ \Sigma Y^2 = \mathrm{SUMY2} \end{array}\right] \text{—— PROGRAM VARIABLE NAMES}$$

Do a summing loop for all N observations where N = # of pairs of observations.

```
      SUMX2=0.
      SUMXY = 0.
      SUMX = 0.
      SUMY = 0.
      SUMY2 = 0.
      DO 10 J=1,N
      SUMXY = SUMXY + X(J)*Y(J)
      SUMX = SUMX + X(J)
      SUMY = SUMY + Y(J)
      SUMX2 = SUMX2 + X(J)**2
   10 SUMY2 = SUMY2 + Y(J)**2
```

3. Calculate r(R), r^2(R2), a(A), b(B) from the formulae (single line Fortran statements).

4. Loop for predicting Y and determining residuals.

```
      DO 20 J=1,N
      YP(J)=A+B*X(J)
   20 RS(J)=Y(J)-YP(J)
```

Where: YP = Y predicted
RS = residuals

5. Find the means and standard deviation of the residuals [RS(J) array]. Standardize the residuals.

```
      DO 30 J=1,N
   30 RZ(J)=(RS(J)-RM)/RSD
```

Where: RZ = standardized residuals array
RM = mean of the residuals RS(J)
RSD = standard deviation of the residuals RS(J)

6. Calculate the F-Ratio from the following formula:

$$F = \frac{r^2(N-2)}{1-r^2}$$

7. Write out information with labels

A. Table of X, Y, YP, RS, RZ

B. Single values of

r(R), r^2(R2),a(A),b(B),F

LESSON 7: DATA MATRICES

Focus:	Using double subscripted arrays for internal data storage
New Statements:	None
Exercise:	Calculating a Z-standardized matrix

Introduction

This lesson introduces the programmer to manipulations required for the analysis of data tables. This involves the use of nested DO loops and implicit DO loops for the reading, writing, and the calculations used in matrix operations.

Double Subscripted Arrays

In computer storage a matrix can be defined as follows.

```
A(1,1),A(1,2),........ A(1,M)
A(2,1),...............   .
   .                     .
   .                     .
   .                     .
   .                     .
A(N,1),............... A(N,M)
```

A single entry of the matrix may be accessed by calling for A(3,6), for example, or for A(I,J), where I and J are previously defined values.

A given row may be worked on by incrementing J while holding I constant and a column may be worked on by incrementing I and holding J constant.

DIMENSION Statement for Double Subscripted Arrays

To reserve storage space in the computer, both the number of rows and the number of columns of the matrix must be dimensioned.

Example:

```
DIMENSION A(20,10),JOHN(20,100)
```

This instruction reserves space for a matrix (designated A) of 20 observations and 10 variables, and a matrix (JOHN) of dimensions 20 by 100. Remember that all subscripted variable data, whether read in or computed, must be dimensioned.

DO Loops and Matrices

Nested DO Loops

Using double subscripted arrays require that two subscripts be incremented using "nested" DO Loops.

Example 1: Adding a constant (2.0) to all the entries of a matrix:

```
     DIMENSION A(20,10)
     DO 10 I=1,20
     DO 10 J=1,10
  10 A(I,J)=A(I,J)+2.0
```

In this example, the computer will add 2.0 to all the entries across the rows (treating one row at a time) because the outer loop will set I (the row number) at 1 and J will be incremented in the inner loop until it reaches 10. In this way, all rows are treated in turn and 2.0 will be added to each entry until the 20th row (I=20) is completed. Note that both DO loops have the same bottom--statement number 10.

Example 2:

```
      DIMENSION A (20,10)
      DO 20 J=1,10
      DO 20 I=1,20
   20 A(I,J)=A(I,J)+2.0
```

In this example, the addition of the constant will occur down the columns. The column will be set at 1, and I (the row) will be incremented to 20. Then J will be set at 2, and I incremented to 20. This continues until the entire matrix is treated.

Summing a Row or Column of a Matrix

Summing a row or a column is similar to adding a constant.

Example:

```
      DIMENSION SUMC(50),A(100,50)
      DO 30 J=1,50
      SUMC(J)=0.0
      DO 30 I=1,100
   30 SUMC(J)=SUMC(J) + A(I,J)
```

Here the column number is set at one in the outer loop. SUMC(J), a new variable representing the sum of the columns, is set at 0.0. In the inner loop, row 1 is designated and the calculation carried out until J = 100. The result is that the first column is summed and that sum is assigned to SUMC(1). Then the computer returns to the outer loop and the second column is worked on, and SUMC(2) is defined. This continues until all the columns have been added and the new vector SUMC(J), defined.

Implicit DO Loops

Regular nested DO Loops cannot be used in READ or WRITE Statements. A special loop has been developed for these cases--an "implicit" DO loop with which the inner loop of the nested DO loops appears in the READ or WRITE Statements.

Examples:

```
    DO 10 I=1,100
 10 READ(5,555)(A(I,J),J=1,50)
555 FORMAT............
```

OR

```
    DO 20 I=1,100
 20 WRITE(6,666)(A(I,J),J=1,50)
666 FORMAT............
```

In these cases the computer would READ or WRITE an entire matrix by scanning across the rows, one at a time.

Exercise

Read in a data table consisting of at least 4 variables and 20 observations. Store the data in a double subscripted array with each column representing one variable. Print out the original data in this form with appropriate labels. Calculate the mean and standard deviation for each column and then standardize each column. Print out the standardized data table along with the mean and standard deviations for each column.

LESSON 8: SUBPROGRAMS

Focus:	Generating and using FUNCTION and SUBROUTINE subprograms
New Statements:	FUNCTION, SUBROUTINE, CALL, RETURN
Exercise:	Calculating a correlation matrix

Introduction

FUNCTION and SUBROUTINE subprograms are complete self-contained Fortran programs that can be used (called) by a main program as needed. FUNCTIONS and SUBROUTINES have the following advantages in structuring large complex programs:

1. Help break a large complicated task into several smaller programs that are used in an orderly fashion by the main program.

2. Saves redundant programming. If a routine (i.e., finding the mean of a vector or sorting a vector) will be used over and over again in a program, the FUNCTION or SUBROUTINE need only be programmed once and called as needed.

3. The programmer can generate a library of commonly used routines or rely on routines available in SUBROUTINE libraries at the local computer center.

<u>General Rules for Generating FUNCTION and SUBROUTINE Subprograms</u>

1. Each FUNCTION or SUBROUTINE is a self-contained program and will contain DIMENSION statements, DO loops, and END statement.

2. The main program makes an implicit call to a FUNCTION and an explicit call to a SUBROUTINE. The main passes information (constants, vectors, arrays) to the subprogram and the subprogram returns value(s) (constant only for a FUNCTION and constants, vectors, or arrays from a SUBROUTINE) which have been generated by the subprogram.

3. In place of the STOP statement (used in the main program), a RETURN statement is used in FUNCTION and SUBROUTINE programs immediately preceding the END statement.

4. FUNCTION and SUBROUTINE elements such as statement numbers, counters, variable names can be used without regard to duplication of the same numbers or names within the main program.

5. All vectors and arrays used in the main program and sent to the FUNCTION or SUBROUTINE must be dimensioned in both the main program and in the FUNCTION or SUBROUTINE.

6. All vectors and arrays generated in the SUBROUTINE and returned to the main program must be dimensioned in both the main program and the SUBROUTINE.

FUNCTION Subprograms

A FUNCTION is a subprogram that is capable of returning a single value stored in the function name. The calling statement in the main program is implicit and has the following form:

A = <u>function name</u> (list of arguments)

The FUNCTION subprogram begins with a FUNCTION statement and ends with RETURN and END statements. A FUNCTION to calculate the mean of a vector of values and a main program illustrating how the FUNCTION is called are shown in the example below.

EXAMPLE:

<u>Main Program</u>

```
      DIMENSION X(100)
C     READ IN 100 OBSERVATIONS
      DO 10 J=1,100
   10 READ(5,11)X(J)
   11 FORMAT (F10.2)
C     CALL FUNCTION XMEAN
      XM=XMEAN(X,100)
C     WRITE OUT MEAN
      WRITE(6,21)XM
   21 FORMAT("1","THE MEAN OF THE VECTOR=",F10.2)
      STOP
      END
```

FUNCTION Subprogram

```
      FUNCTION XMEAN(A,N)
      DIMENSION A(100)
      XN=N
      SUM=0.
      DO 10 J=1,N
   10 SUM=SUM+A(J)
      XMEAN=SUM/XN
      RETURN
      END
```

NOTE: 1. The name assigned to the FUNCTION must be in accord with the naming convention for variable names (i.e., must begin with an alphabetic character denoting floating point or integer mode and contain no more than 6 symbols).

2. The calling arguments, or arguments sent to the FUNCTION do not need to agree in name with the arguments listed in the FUNCTION Statement, but must agree in type (integer or floating point).

3. In a card deck, the usual placement of the FUNCTION cards would be immediately after the main program cards. No special JCL is necessary.

SUBROUTINE Subprograms

A SUBROUTINE is a subprogram that is capable of returning any number of single values and/or vectors of values and/or arrays of values to the main program. The calling statement in the main program is explicit and has the following form:

CALL subroutine name (list of arguments)

The SUBROUTINE subprogram begins with a SUBROUTINE Statement and ends with RETURN, END Statements. A SUBROUTINE to calculate the mean and standard deviation of a vector of values and a main program illustrating how the SUBROUTINE is called are shown in the next example.

Example:

Main Program

```
      DIMENSION X(100)
C     READ IN 100 OBSERVATIONS
      DO 10 J=1,100
   10 READ(5,11)X(J)
   11 FORMAT(F10.2)
C     CALL SUBROUTINE STATS
      CALL STATS(X,100,XM,SX)
C     WRITE OUT MEAN AND STANDARD DEVIATION
      WRITE(6,21)XM,SX
   21 FORMAT("1","MEAN=",F10.2/1X,"STANDARD
         DEVIATION= ",F10.2)
      STOP
      END
```

SUBROUTINE

```
      SUBROUTINE STATS(A,N,XMEAN,STDEV)
      DIMENSION A(100)
      XN=N
      SUMX=0.
      SUMX2=0.
      DO 10 J=1,N
      SUMX=SUMX+A(J)
   10 SUMX2=SUMX2+A(J)**2
      XMEAN=SUMX/XN
      STDEV=SQRT(SUMX2/XN-XMEAN**2)
      RETURN
      END
```

NOTE:

1. In the CALL statement X and 100 are the "sending arguments" and XM and XS are the "returning arguments." These arguments need not be in any particular order but the order of the list of arguments in the SUBROUTINE statement must match the order of the list of arguments in the CALL statement.

2. The arguments in the CALL statement do not need to be named the same as in the SUBROUTINE statement. However, they must match in type (integer and floating point) and must have the equivalent dimension of the vectors or arrays used in the main program.

Exercise

Using some simple matrix manipulations, the standardized matrix derived in exercise 7 can be used to calculate a correlation matrix of linear correlation coefficients between all pairs of variables. The mathematical notation is shown below and the student will need only to use the correct sequence of matrix manipulation subroutines (Appendix C) to program the mathematics. Make this addition to the program developed in Exercise 7 and write out the results with appropriate labels.

<u>Calculation of a correlation matrix</u>*

$$R = \frac{1}{n-1} \cdot Z^T Z$$

where R = correlation matrix

Z = standardized matrix

Z^T = standardized matrix transposed

n = number of observations

*Reference: <u>An Introduction to Statistical Models in Geology</u>
W. C. Krumbein and F. A. Graybill
McGRAW HILL BOOK CO. N.Y. 1965, pp. 383-391.

SECTION II

FORTRAN PROGRAMMING OF SPATIAL DATA

INTRODUCTION

These four lessons constitute a broad introduction to the programming of spatial data. The intent of this section is to show geography students the usefulness of various aspects of machine processing. The lessons cover the numerical methods of a simple linear trend surface; the use of a line printer for mapping; an introduction to electronic digitization and the associated calculations of areas and perimeters; and the basic programming of a line plotter for drawing map outlines.

Included in the lessons is some advanced FORTRAN programming. Not all of the FORTRAN presented in each lesson may be directly applicable to the exercise, but it is organized in useful groups of techniques. In some cases the FORTRAN is an extended review of methods which were introduced in the first eight lessons.

LESSON 9: DATA MANIPULATION

Focus:	Flags, variable formats; internal data generation, and DATA statements.
New Statement:	DATA
Exercise:	Programming a linear trend surface.

Introduction

This lesson introduces methods for reading in data using a flag, variable formats, techniques for generating data internally, and the utility of DATA statements. All of these data-handling procedures are aimed at providing a means through which programs can be made more general or "universal."

Reading in Data

Programming to Read in Data Using a Flag

A user often has a large data set but does not know the exact number of observations. A flag can be placed on the last card and the data read in with the number of observations (cards) automatically counted. The flag is simply a number greater than zero which is punched only on the last card. The program on the next page will read up to 700 cards terminating the read when it encounters a card with a number greater than zero punched in column 80.

```
      DIMENSION X(700)
C     INITIALIZE COUNTER AND START READ LOOP
      I=1
   10 READ(5,11) X(I),NFLAG
   11 FORMAT(F10.2,69X,I1)
      IF(NFLAG.GT.0) TO TO 20
      I=I+1
      GO TO 10
   20 NOBS=I
C     COMPLETION OF READ LOOP AND COUNTING OF
C      CARDS (NOBS)
```

Another type of flag is discussed under Logical Formating in Lesson 10.

Variable Format:

Often the data to be read into a general program is not in the same format as specified for the READ statement in the program. To overcome this problem, a format can be read in from the data deck and then used to read in the data.

```
      DIMENSION X(500), FORM1(20)
C     READ IN # OF OBSERVATIONS
      READ(5,11) NOBS
   11 FORMAT(I3)
C     READ IN DATA FORMAT
      READ(5,21) (FORM1(J),J=1,20)
   21 FORMAT(20A4)
C     READ IN DATA ACCORDING TO FORMAT
C     STORED IN ARRAY FORM1
      DO 30 J=1,NOBS
   30 READ(5,FORM1) X(J)
C     CONTINUE REST OF PROGRAM
```

The first three cards of the data deck would look like this:

```
 COLS 123456789
CARD1  30        ←NO. OF OBS.
CARD2 (F10.2)    ←FORMAT
CARD3 19.613     ←FIRST DATA VALUE
```

Generating Data Internally

Often the programmer needs to design a program which will generate its own data internally rather than have it read in. This is particularly true of programs which produce tables of information. As a simple example, suppose we wished to generate a table of squares, cubes, square roots, cube roots, reciprocals and logs (base 10) of integer values ranging from 1-100. The program calculating portion is done in two parts:

1. Generate a vector of numbers

2. Generate new vectors of the square, cube, square root, etc., of the values in the first vector.

```
      DIMENSION NX(100),XS(100),XC(100),XSR(100)
     * ,XCR(100),RX(100),XL(100)
C     GENERATE THE VALUES 1-100 AND
C     STORE IN ARRAY NX (INTEGER VECTOR)
      DO 10 J=1,100
   10 NX(J)=J
C     GENERATE THE TABLE VALUES
      DO 20 J=1,100
      X=NX(J)
      XS(J)=X**2
      XC(J)=X**3
      XSR(J)=SQRT(X)
      XCR(J)=X**(1./3.)
      RX(J)=1./X
   20 XL(J)=ALOG10(X)
C     WRITE HEADER FOR TABLE AND OUTPUT 50 VALUES TO A PAGE
      DO 50 KK=1,2
      WRITE (6,31)
   31 FORMAT("1",///1X,"  X-VALUE       SQUARE        CUBE"
     1 ,T40,"SQUARE ROOT   CUBE ROOOT   RECIPROCAL     LOG(10)")
      N1=(KK-1)*50+1
      N2=KK*50
      DO 40 J=N1,N2
   40 WRITE(6,41)NX(J),XS(J),XC(J),XSR(J),XCR(J),RX(J),XL(J)
   41 FORMAT(6X,I4,8X,F6.0,5X,F8.0,5X,F7.3,6X,F7.3,6X,F6.4,8X,F5.3)
   50 CONTINUE
      STOP
      END
```

If you wish to expand this program to make tables of functions of numbers beyond 1-100 (i.e., for 1-1000), do the following (note that the current program will only operate on 100 numbers at a time, therefore all we need to do is loop the current program 1000/100 or 10 times.):

```
      DIMENSION NX(100),XS(100),XC(100),XSR(100),
     *XCR(100),RX(100),XL(100)
C     START MAIN PROGRAM LOOP
      DO 50 JJ=1,10
C     GENERATE 100 VALUES 1-100  FOR JJ=1, 101-200 FOR
C        JJ=2,ETC. AND STORE IN NX ARRAY (INTEGER VECTOR)
      DO 10 J=1,100
   10 NX(J)=(JJ-1)*100+J
C     GENERATE THE TABLE VALUES
        ↑
        Exactly the same as the above program.
        ↓
   50 CONTINUE
      STOP
      END
```

DATA Statements

Values such as constants or symbols may be established for the program without reading in the values as symbols from data cards or using an algebraic expression to set a variable name equal to some value. This is accomplished through the use of a DATA Statement.

Example 1:

Setting constants equal to some value.

```
COL 7
    DATA HIT/7./
    DATA  IT/7/
```

The variable name HIT will have the value 7. assigned to it and the variable name IT will have the value 7 assigned to it.

Example 2:

Setting a vector of constants equal to a series of values.

```
DIMENSION FIT(5),NIT(5)
DATA FIT/7.,8.,9.5,2.1,3.0/
DATA NIT/7,8,9,21,30/
```

FIT(1) through FIT(5) will be assigned the values 7. through 3.0 and NIT(1) through NIT(5) will be assigned the values 7 through 30.

Example 3:

Setting a variable equal to some symbol.

```
DATA AST/"*"/
```

The symbol is placed between quotes.

Example 4:

Setting a vector equal to a series of symbols.

```
DIMENSION ISX(5)
DATA ISX/"*","+","A","2","#"/
```

Each symbol is placed between the quotes and separated by a comma.

Example 5:

Setting a matrix equal to some values

$$A=\begin{bmatrix} 2.1 & 3.4 & 5.6 \\ 7.8 & 9.2 & 11.1 \\ 13.2 & 4.3 & 1.1 \\ 6.2 & 3.9 & 9.6 \end{bmatrix}$$

The matrix (A) of numbers can be placed in a data statement by writing the matrix as a vector containing the columns one behind each other.

$$\text{DIMENSION A(4,3)}$$
$$\text{DATA A/}\underbrace{\text{2.1,7.8,13.2,6.2}}_{\text{Col 1}}\text{,}\underbrace{\text{3.4,9.2,4.3,3.9}}_{\text{Col 2}}\text{,}\underbrace{\text{5.6,11.1,1.1,9.6}}_{\text{Col 3}}\text{/}$$

Exercise

Develop a linear trend surface program from the guide given in the next several pages.

The necessary SUBROUTINE programs for the matrix operations are given in Appendix C and a review of the mathematics involved is given in Appendix D. A data set consisting of the precipitation values for 100 climate stations for Illinois is given in map form (Figure 9-1).

LINEAR TREND SURFACE PROGRAM
(GENERAL STRUCTURE)

1. Read in a title card 1-80 cols and store in a vector ID (20A4).

2. Loop to read in one (1) dependent variable (Z) and two (2) independent variables (X,Y) per card (observation). Store each in a vector, i.e., Z(J), X(J),Y(J) for J=1,NOBS.

3. Generate the necessary sums for the S-Matrix of coefficients and fill in S-Matrix. (For a review of the math operations see Appendix D).

$$S = \begin{bmatrix} N & \Sigma X & \Sigma Y \\ \Sigma X & \Sigma X^2 & \Sigma XY \\ \Sigma Y & \Sigma XY & \Sigma Y^2 \end{bmatrix} \qquad \text{where } \Sigma = \sum_{J=1}^{NOBS}$$

From the S-Matrix shown above five (5) sums are required.

```
C     SET SUMS=0
      SX=0.
      SY=0.
      SXY=0.
      SX2=0.
      SY2=0.
C     CALCULATE SUMS
      DO 10 J=1,NOBS
      SX=SX+X(J)
      SY=SY+Y(J)
      SXY=SXY+X(J)*Y(J)
      SX2=SX2+X(J)**2
   10 SY2=SY2+Y(J)**2
C     FILL IN S-MATRIX
      S(1,1)=NOBS
      S(1,2)=SX
```

```
      S(1,3)=SY
      S(2,1)=SX
      S(2,2)=SX2
      S(2,3)=SXY
      S(3,1)=SY
      S(3,2)=SXY
      S(3,3)=SY2
```

4. Generate the necessary sums and fill in g vector.

Where $g = \begin{bmatrix} \Sigma Z \\ \Sigma ZX \\ \Sigma ZY \end{bmatrix}$

```
C     SET SUMS=0
      SZ=0.
      SZX=0.
      SZY=0.
C     CALCULATE SUMS
      DO 20 J=1,NOBS
      SZ=SZ+Z(J)
      SZX=SZX+Z(J) *X(J)
   20 SZY=SZY+Z(J)*Y(J)
C     FILL IN G VECTOR USING 4TH COL OF S MATRIX
      S(1,4)=SZ
      S(2,4)=SZX
      S(3,4)=SZY
```

5. Solve the set of simultaneous equations (see Appendix C).

```
C     CALL EQUAT TO SOLVE EQUATIONS
      CALL EQUAT(S,3,4)
```

6. Calculate predicted Z-values and residuals and store in vectors, i.e., ZP(J),RZ(J).

```
C     CALCULATE ZP AND RESIDUAL VECTOR
      DO 30 J-1,NOBS
      ZP(J)=S(1,4)+S(2,4)*X(J)+S(3,4)*Y(J)
   30 RZ(J)=Z(J)-ZP(J)
```

7. Calculate standardized residuals.

```
C     SET SUMS =0.
      SRZ=0.
      SRZ2=0.
      DO 40 J=1,NOBS
      SRZ=SRZ+RZ(J)
   40 SRZ2=SRZ2+RZ(J)**2
C     CALCULATE MEAN AND STANDARD DEVIATION OF RESIDUALS
      RZM=SRZ/NOBS
      RZS=SQRT(SRZ2/NOBS-RZM**2)
      STANDARDIZE RESIDUALS
      DO 50 J=1,NOBS
   50 RS(J)=(RZ(J)-RZM)/RZS
```

8. Write out the following in a table with labels:

X(J), Y(J), Z(J), ZP(J), RZ(J), RS(J).

9. Calculate the variation measures.

```
C     CALCULATE TOTAL VARIATION
C     UNEXPLAINED VARIATION WAS PREVIOUSLY CALCULATED
C     AS THE SUM OF THE RESIDUALS SQUARED (SRZ2)
C     THE EXPLAINED VARIATION = TOTAL VARIATION - UNEXPLAINED VARIATION
      ZMEAN=SZ/XNOBS
      SUMT=0.
      DO 60 J=1,NOBS
   60 SUMT=SUMT+(ZMEAN-Z(J))**2
      EXP=SUMT-SRZ2
C     CALCULATE R AND R2
      R2=EXP/SUMT
      R=SQRT(R2)
C     CALCULATE F
      F=(R2*( NOBS-3.))/(2*(1.-R2))
```

10. Write out variation measures.

Write out, R, R2 and F with labels.

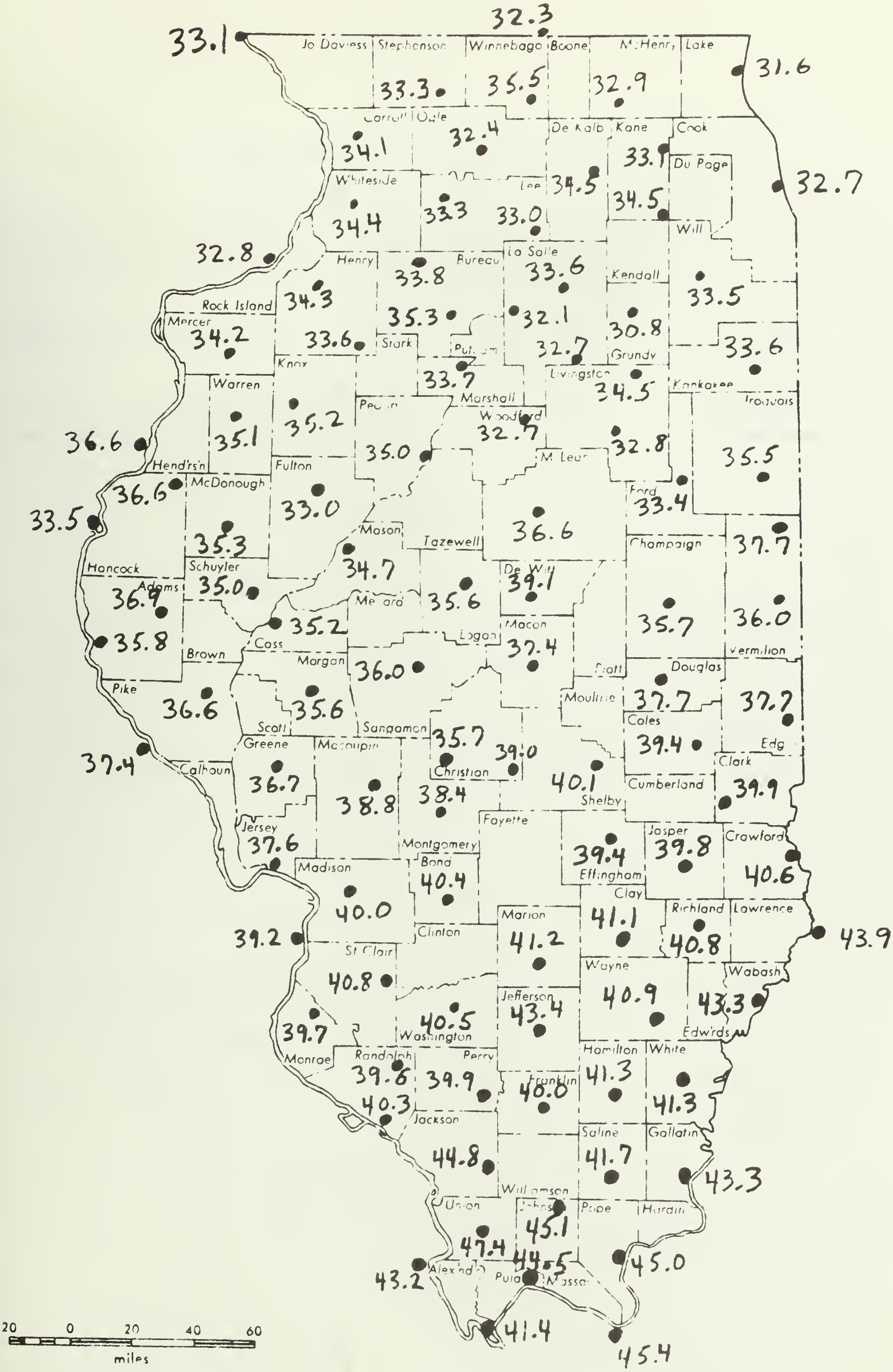


Fig. 9-1 Illinois Annual Precipitation (inches)

LESSON 10: MISCELLANEOUS TOPICS

Focus:	Type specification, further formatting, additional branching statements, double precision and line printed maps.
New Statements:	INTEGER, REAL, LOGICAL, IF computed GO TO, DOUBLE PRECISION
Exercise:	Mapping subroutine for 1st degree trend surface

Introduction

This lesson consists of miscellaneous topics that are useful or necessary aspects of the FORTRAN language needed for advanced programming. An attempt should be made by the programmer at this stage to incorporate these statements in order to develop breadth in his or her programming capability. The exercise, a line printed mapping subroutine, introduces the student to a fairly complex program which makes use of some of these new statements.

Type Specification

Type specification statements are used to indicate variable types. Type specification overrides the naming convention of A-H, O-Z first letter for floating point and I-N first letter for integer. The general form for type declaration is:

type*n, variable name, variable name,......

where <u>type</u> may be INTEGER
REAL
LOGICAL
COMPLEX

n is the size of the variable in bytes

The size (n) is usually omitted: INTEGER, REAL and LOGICAL default to *4 and COMPLEX to *8. If n is specified, options other than *4 may be REAL*8, INTEGER*2, LOGICAL*1, and COMPLEX*16

variable name is the single variable name, vector, or array to be specified as a particular type.

Examples:

```
REAL A,B,N,
INTEGER I,J,X,Y,Z
```

Further Formatting

	Formats	General Form
Previously Discussed	Floating Point	Fa.b
	Integer	Ia
	Alphanumeric	Aa
Added	Exponential	Ea.b
	General	Ga.b
	Logical	La

Where:

a = field width

b = decimal positions

Exponential Example

The number 123456789000 can be written in exponential notation as $.123456789X10^{12}$, $1.23456789X10^{11}$, $12.3456789X10^{10}$ etc. To write out this number on the computer the following formats could be used:

```
E14.8 producing .12345678E+12
E14.7 producing 1.2345678E+11
E14.6 producing 12.345678E+10
```

Note that the six columns of the field width are reserved for the decimal point, the four characters of E notation (i.e. E+10) and the negative sign (if required).

General Example

A general format specification may be available on some computers allowing for the substitution of a G format for F, I, E and L formats. To read an integer, floating point and exponential number from a data card a G format can be used with all three values as shown below.

```
COL 1          10          20          30
                 ↓           ↓           ↓
               986      736487      1.2E+7

   READ(5,11)IVAL,FVAL,EVAL
11 FORMAT(3G10.3)
```

Then:

IVAL = 986
FVAL = 736.487
EVAL = $1.2X10^{7}$

Logical Example

A variable may be declared logical (see type declaration section) and can assume one of two values--true or false. For example, a variable LASTPT can be declared logical in the following ways:

```
LOGICAL*1 LASTPT
LOGICAL*4 LASTPT
LOGICAL LASTPT
```

If declared LOGICAL*1, the value T or F (for True or False) can be

assigned to LASTPT. If declared LOGICAL*4, words beginning with T or F (for True or False) can be assigned to LASTPT.

The logical variable can be used as a flag in reading in data and provides a method for counting the number of cards read. The program below shows the use of a logical flag.

```
      DIMENSION X(100)
      LOGICAL*1 LASTPT
      I=1
   10 READ(5,11) X(I), LASTPT
   11 FORMAT(F10,2,69X,L1)
      IF(LASTPT) GO TO 20
      I=I+1
      GO TO 10
   20 NOBS=I
```

A T punched in column 80 of only the last data card would flag the end of the data set. On all other cards, LASTPT would read a blank and would assume a value of False.

Branching Statements

In previous lessons we learned the use of a logical IF Statement coupled with the unconditional branching GO TO statement to provide a mechanism for conditional branching. A typical example would be:

```
IF(A.GT.B.AND.C.LT.D) GO TO 10
```

Several other statements exist which provide conditional branching. These are discussed below.

IF Statement

(This is not a logical IF Statement.) The general form of the IF statement acts like three GO TO statements.

IF(EXPRESSION)#,#,#

statement # for positive result of expression
statement # for zero result of expression
statement # for negative result of expression
constant or operation between several variables

Example:

IF(A-B) 10,21,32

If the result of A-B is negative, the program will branch (GO TO statement number 10; if the result of A-B is zero, the program will branch to statement number 21; and if the result is positive, the program will branch to statement number 32.

Single value variables (constants) may be used alone:

IF(A) 10,21,32

will produce the same results for a negative, zero, and positive A.

Computed GO TO

The general form of the computed GO TO is:

GO TO (#,#,#,#,#),K ← integer position indicator

statement # for K=5
statement # for K=4
statement # for K=3
statement # for K=2
statement # for K=1

Example:

GO TO (111,93,92,22,34),K

If K=3 the program will branch (GO TO) to statement #92. Often the integer position indicator is a calculated value when using a computed GO TO. For example, if the following statements occurred in a program, the results would be those shown in the table.

```
L=X/100 +1
GO TO (11,21,31,41,51),L
```

If X is Between	The program will branch to (GO TO) statement #
0 - 100	11
100 - 200	21
200 - 300	31
300 - 400	41
400 - 500	51

Double Precision

DOUBLE PRECISION is a floating point variable type with which numbers may be stored in a longer memory location, thus allowing for greater precision in the digital representation of a number. Singe precision values (the automatic default for a named floating point variable) will have between 7 and 11 significant digits stored, depending on the type of computer. DOUBLE PRECISION will allow between 14 and 25 digits to be stored. The rules for using DOUBLE PRECISION are:

1. Type declaration:

   DOUBLE PRECISION X,Y,Z,

   or REAL*8 X,Y,Z

2. In a FORTRAN algebraic expression, if the left-hand side is declared a DOUBLE PRECISION variable, the computer will temporarily convert those right-hand side variables <u>not declared</u> DOUBLE PRECISION <u>to</u> DOUBLE PRECISION.

3. If a FORTRAN supplied function (SQRT, TAN) is being used with arguments that have been declared DOUBLE PRECISION, then the function must be type declared as a DOUBLE PRECISION usually by preceding the function name with the letter D.

```
REAL*8 X,Y
X=SQRT(Y)        {not allowed

REAL*8 X,Y
X=DSQRT(Y)       {allowed
```

Line Printed Maps

Crude but useful maps of spatial variables can be made on the line printer. This section will introduce you to the principles of line printer mapping, using a first degree trend surface as an example.

A source map of such as the one from Exercise 9 may be used to fit a 1st degree trend surface ($Z=a_0+a_1X+a_2Y$) to a sample of data points. The equation can then be evaluated for any number of points by simply substituting the requisite X,Y coordinates and solving for Z. If a large number of systematically chosen points are evaluated, we can map the distribution using symbols to represent the Z value class into which each point falls.

The process is quite simple. Starting with a source map and set of sample points, fit a surface $Z = a_0 + a_1X + a_2Y$ to the sample points.

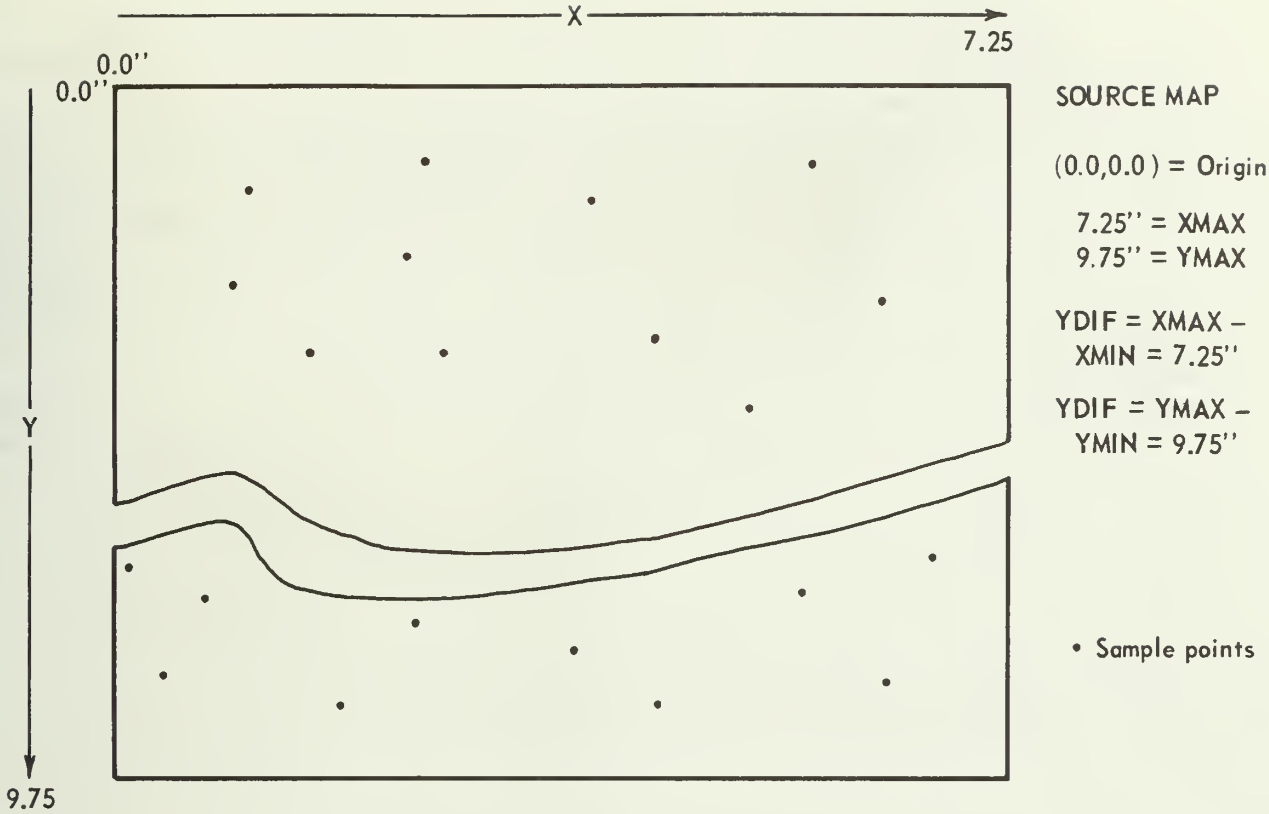


Fig. 10-1 Source Map Dimensions

The equation is then systematically evaluated for a large number of points corresponding to printing positions on the computer lister sheet as follows. Each printing position has a corresponding X,Y position on the original source map. The X,Y coordinates are substituted into the trend surface equation and a Z value calculated. A symbol is assigned to the print position according to the class or level within which the calculated Z value falls.

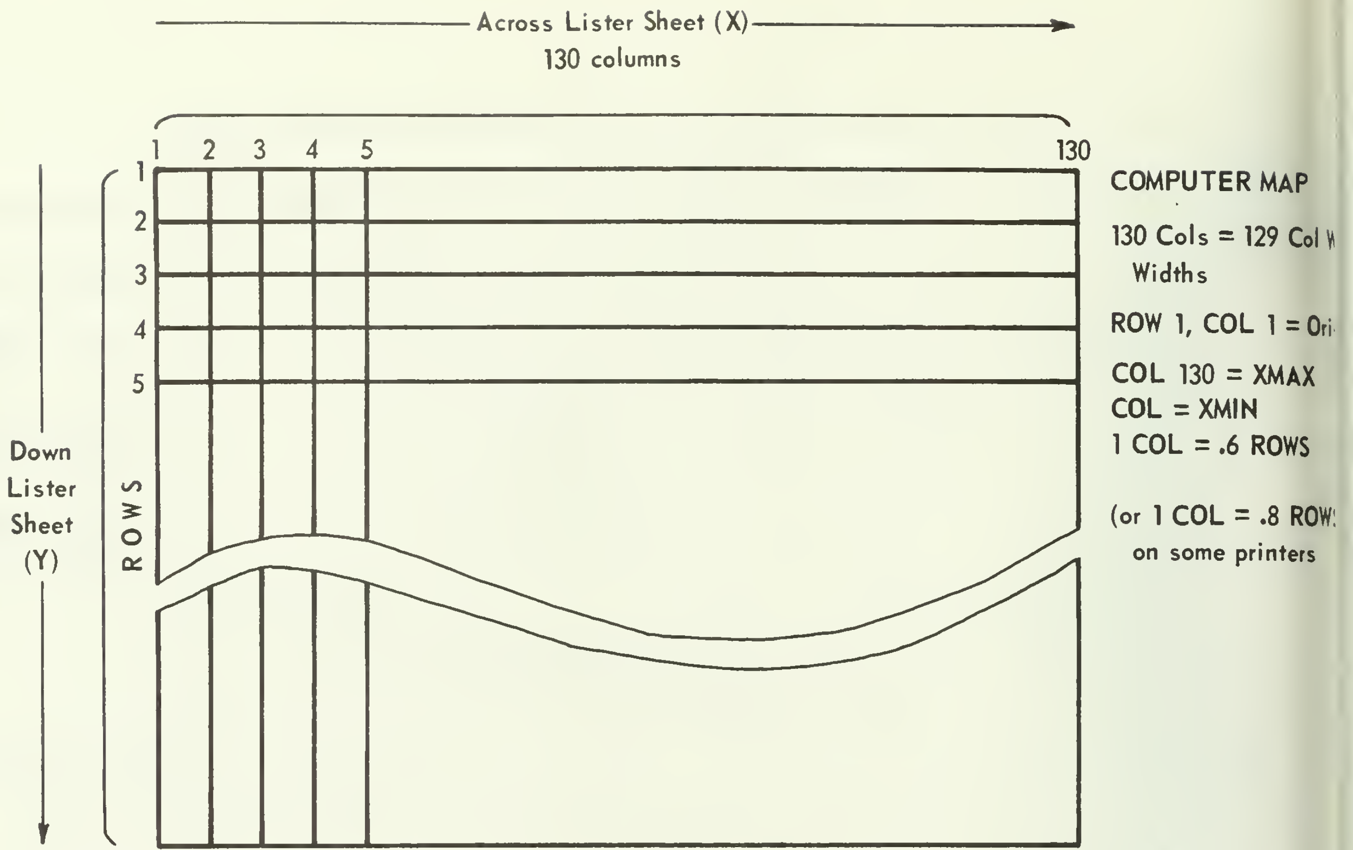


Fig. 10-2 Output Map Dimensions

To produce the computer map, the relationships indicating the correspondence between the source map and computer map need to be calculated. This involves two steps: scaling and positioning. The third step is map production or printing.

Scaling

If we let the map width (X) for the computer map = 130 COLS or 129 COL WIDTHS, the map length (Y) is $\frac{\text{YDIF}}{\text{YDIF}} = \frac{9.75}{7.25} = 1.345$ X MAP WIDTH.

Since there are 10 COLS per inch across on the printout and 6 ROWS per inch down on the printout then the map length in ROWS is:

$$1.345 \text{ X } 130. \text{ X } \frac{6}{10} = 104.91 \text{ ROWS}$$

Or the number of ROW WIDTHS (RW)

$= 104.91$ ROWS -1

$= 103.92$ ROW WIDTHS

Since we need an integer # of ROW WIDTHS, the floating point # of ROW WIDTHS above is rounded and truncated by converting to an integer value:

$$\begin{aligned} \text{NROWS} &= \text{RW} + .5 \\ &= 104.91 + .5 \\ &= 105 \text{ (Truncated)} \end{aligned}$$

Positioning

Now we need to determine the position of ROW 1, COL 1 (computer map) on the source map. These map coordinates are then used in $Z = a_0+a_1X+a_2Y$ to determine the Z at ROW 1, COL 1 of the computer map. If ROW 1, COL 1 on the computer map is considered the origin, the coordinates on the source map are YMIN, XMIN or (0,0). To determine the position of ROW 1, COL 2 (computer map) on the source map, an X increment is needed. 129 COL WIDTHS on the computer map is equal to 7.25 inches on the source map. 1 COL WIDTH on the computer map is equal to 7.25 inches on the source map. 1 COL WIDTH on the computer map $= \frac{7.25''}{129} = .056'' = \text{XI}$.

The coordinates on the source map for ROW 1 = YMIN. The coordinates on the source map for COL 2 = XMIN + XI, or = 0 + .056 = .056''.

The coordinates on the source map corresponding to ROW 1, COL 2 on the computer map are (0,.056). These are entered into $Z = a_0+a_1X+a_2Y$ to determine Z. As a final example, consider the location COL 2, ROW 3 on the computer map.

ROW 3 will have a location of XMIN + 2*XI = .112'' on the source map.

COL 2 will have a coordinate location of YMIN + YI, where YI is determined as follows:

$$YI = \frac{9.75''}{\#\ ROWS} = \frac{9.75''}{105} = .093''$$

$$\begin{aligned} COL\ 2 &= YMIN + YI \\ &= 0 + .093'' \\ &= .093 \end{aligned}$$

The source map coordinates for COL 2, ROW 3 = (.112, .093). Z will then be:

$$Z = a_o + a_1(.112) + a_2(.093)$$

Mapping

Once the Z has been determined for a print position on the computer map, the data class it falls into must be determined. A reference contour is determined (e.g., 40'' for the Illinois precipitation map of Exercise 9) and a contour interval chosen (e.g., 2''). The map will then have contour lines of:

.......34'', 36'', 38'', 40'', 42'', 44'', 46''.......

If our program has symbols to represent 21 classes such as:

(#11)

A, B, C, D, E, F, G, H, I, J, $, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0

and we number them from left to right, then the $ symbol is the eleventh symbol. Then any calculated Z value can be assigned a symbol by:

$$NSP = \frac{Z - REF}{CON} + 11$$

here the eleventh symbol is always the reference contour and NSP is the number left or right of the symbol.

For a calculated Z value of 39.9''

$$NSP = \frac{39.9-40.}{2} + 11 = \frac{-.1}{2} + 11$$

$$= 11-.05$$

$$= 10.95$$

$$= 10 \text{ (Truncated)}$$

Symbol #10 = J

For a calculated Z value of 40.0''

$$NSP = \frac{40.0-40.}{2} + 11 = 0+11$$

$$= 11$$

Symbol #11 = $

For a calculated Z value of 41.9''

$$NSP = \frac{41.9-40.}{2} + 11 = \frac{1.9}{2} + 11$$

$$= .95 + 11$$

$$= 11.95$$

$$= 11 \text{ (Truncated)}$$

Symbol #11 = $

For values of Z all between 40.0'' and 41.999''. NSP = 11 corresponding to the $ symbol. The class (10) represented by symbol J has values of 38.0'' - 39.999''.

Use a data statement to assign the symbols to a vector ISX as shown below.

DATA ISX/''A'', ''B'', ''C'', ''D'', ''E'', ''F'', ''G'', ''H'', ''I'', ''J'', ''$'', ''1'', ''2'', ''3'', ''4'', ''5'', ''6'', ''7'', ''8'', ''9'', ''0''/

A vector representing one row of the map can be filled with symbols defined by ISX(NSP). The row can then be printed and another row filled

and printed until the map is complete. SUBROUTINE MAPT on the next page is a 1st degree trend surface mapping program.

Exercise

Add SUBROUTINE MAPT to the program developed in Lesson 9. The CALL statement should be placed just before the STOP, END Statements. The call is as follows:

CALL MAPT (C,ID)

Where C = vector of coefficients

ID = title vector

<u>NOTE</u>: MAPT reads in control card of floating point numbers

```
COLS   1-10 XMIN OF SOURCE MAP
      11-20 XMAX OF SOURCE MAP
      21-30 YMIN OF SOURCE MAP
      31-40 YMAX OF SOURCE MAP
      41-50 REFERENCE CONTOUR
      51-60 CONTOUR INTERVAL
         61 LINE PRINTING (6 lines per inch or 8 lines per inch)
```

```
      SUBROUTINE MAPT(C,ID)
      DIMENSION C(3),MAP(130),ISX(45),ID(20),NCOUNT(45)
      REAL*8 AX,AY
      DATA ISX/"-"," ","J"," ","I"," ","H"," ","G"," ","F"," ","E",
     1 " ","D"," ","C"," ","B"," ","A"," ","$"," ","1"," ","2"," ",
     2 "3"," ","4"," ","5"," ","6"," ","8"," ","9"," ","0"," ","*"/
      DATA MINUS/"-"/
C     READ MAP PARAMETERS FROM SOUCE MAP
      READ(5,11)XMIN,XMAX,YMIN,YMAX,REF,CON,LINPR
   11 FORMAT (6F10.2,I1)
C     SET LISTER SHEET MAP PARAMETERS
      XDIF=XMAX-XMIN
      YDIF=YMAX-YMIN
      NX=130
      NDEG=1
      XLP=LINPR/10.
      XNY=(YDIF/XDIF*130.)*XLP+.5
      XI=XDIF/129.
      YI=YDIF/(XNY-1.)
      NY=XNY
C     WRITE HEADER AND MAP INFORMATION
      WRITE(6,31) (ID(J),J=1,20)
   31 FORMAT("1",20A4,////)
```

```
      WRITE(6,32)XMIN,XMAX,YMIN,YMAX,NDEG,REF,CON
  32  FORMAT(1X,"XMIN= ",F10.2,/1X,"XMAX= ",F10.2,/1X,"YMIN= ",
     1 F10.2,/1X, "YMAX= ",F10.2,/1X, "DEGREE OF FIT= ",I2,/1X
     2 ,"REFERENCE CONTOUR= ",F10.1,/1X,"CONTOUR INTERVAL= ",
     3 F10.1,////)
C     SET COUNTER TO ZERO
      DO 35 I=1,45
  35  NCOUNT(I)=0
C     WRITE OUT TOP BORDER
      WRITE(6,41)(MINUS,J=1,130)
  41  FORMAT(1X,130A1)
C     START LOOP TO GENERATE MAP
      DO 60 JJ=1,NY
  40  AY=YMIN+(JJ-1)*YI
      DO 50 J=1,NX
      AX=YMIN+(J-1)*XI
C     CALCULATE Z-VALUE
      ZZ=C(1)+C(2)*AX+C(3)*AY
C     DETERMINE SYMBOL # IN ISX VECTOR
      NSP=(ZZ-REF)/CON+23.
C     CHECK FOR LIMITS ON NSP
      IF(NSP.LT.1) NSP=1
      IF(NSP.GT.45) NSP=45
C     COUNT EACH SYMBOL
      NCOUNT(NSP)=NCOUNT(NSP)+1
C     FILL IN ONE ROW OF MAP
  50  MAP(J)=ISX(NSP)
C     WRITE OUT ONE ROW OF MAP
      WRITE(6,41) (MAP(J),J=1,130)
  60  CONTINUE
C     WRITE OUT BOTTOM OF MAP BORDER
      WRITE(6,41)(MINUS,J=1,130)
      CALL LEGEND(NCOUNT,ISX,REF,CON)
      RETURN
      END
      SUBROUTINE LEGEND(NCOUNT,ISX,REF,CON)
      DIMENSION NCOUNT(45),ISX(45)
      J = 1
  10  IF(NCOUNT(J) .GT. 0)GO TO 20
      J=J+1
      GO TO 10
  20  MIN=J
      K=45
  30  IF(NCOUNT(K) .GT. 0) GO TO 40
      K=K-1
      GO TO 30
  40  MAX=K
      WRITE(6,41)
  41  FORMAT(1X,"SYMBOL",10X,"CLASS",10X,"FREQUENCY"/)
      DO 100 I=MIN,MAX
      XI=I
      XLCV=REF+(XI-23.)*CON
      XUCV=XLCV+CON
 100  WRITE(6,111) ISX(I),ISX(I),ISX(I),ISX(I),XLCV,XUCV,NCOUNT(I)
 111  FORMAT(2X,4A1,7X,F6.1,"-",F6.1,6X,I7,///)
      RETURN
      END
```

LESSON 11: SPATIAL DATA SETS

Focus:	Digitizing and manipulating spatial data sets.
New Statements:	None
Exercise:	Perimeter and area of a closed figure from digitized coordinates.

Introduction

Spatial locations as determined by Cartesian coordinates are more readily and accurately obtained with the use of a digitizer. This lessom introduces students to electronic digitizing and some fundamental manipulations of spatial data.

Digitizing

A sonic digitizer

A digitizer pen is shown in Figure 11-1. This particular instrument (Graf-Pen) operates on a sonic principle using a noise generating pen and two microphones to pick up the pulses. Figure 11-2 shows the sequence for digitizing a single point. The steps are as follows:

1. The pen is placed at the desired point and pressed down.

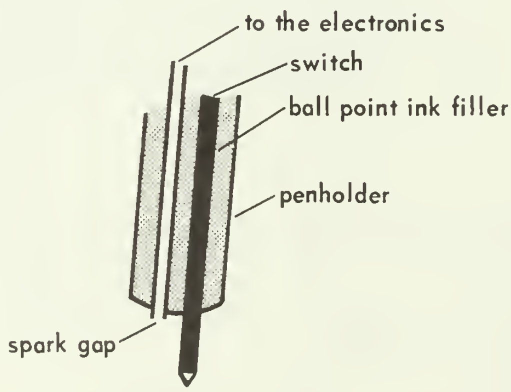


Fig. 11-1 Digitizer Pen

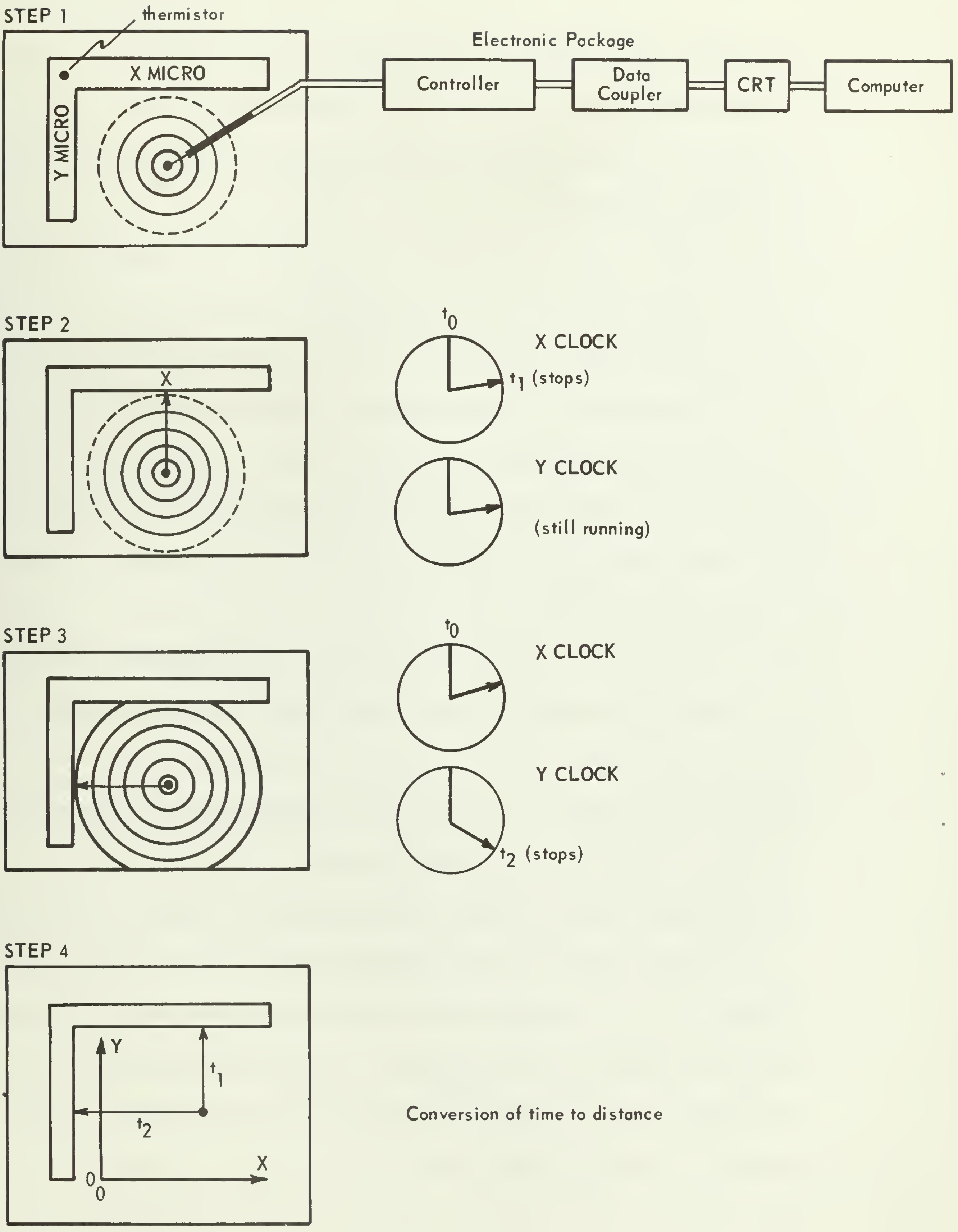


Fig. 11-2 Digitizing Sequence

This activates a switch which causes a current to jump the spark gap creating a high frequency noise which then travels toward the microphones. The activated switch also starts two clocks in the electronics package at the same time the spark initiates.

2. The wave front initiated by the spark travels through the air (velocity of sound is dependent on the temperature of the air and changes due to fluctuating temperatures are compensated for by a thermistor incorporated in the digitizer arms) and eventually impinges on the X microphone causing the X clock to stop. The elapsed time is used to calculate the distance the sound travels and hence the X-coordinate. The Y clock continues to run.

3. The wave front eventually reaches the Y microphone and stops the Y clock. The elapsed time is used to calculate the Y-coordinate.

4. For this particular system seven coordinate pairs are stored on a line in the following format:

/0050,0063/0053,0068/0046,0119/0074,0089/0121,0091/0009,0023/0360,0295

The first coordinate pair /0050,0063 is X=.50 inches and Y=.63 inches. After each line is filled, the line is sent to the computer by a carriage return initiated by the Graf-Pen controller. The controller also determines the Cartesian configuration for the digitizer (origin, size, etc.).

Entering Z-values

Usually the location (X,Y coordinates) and some data value (Z-value) associated with that particular location are desired. The digitizer can only be used to obtain the X,Y coordinates and perhaps indirectly the Z-values. There are three basic alternatives for entering Z-values as part of the data file.

1. The Z-values are entered on the CRT terminal after the X,Y coordinates are obtained. The Z-values must be in the same sequence as the X,Y coordinates. This involves numbering the points on the source map and being extremely careful not to get out of order when digitizing or entering Z-values. For small data sets (definitely under 100 points) this is the simplest approach.
2. A special pen or cursor may be obtained which has a key board (usually only numeric) for entering the Z-value after the coordinates have been obtained. This setup usually requires a more expensive controller and may present some formatting problems on some systems.
3. A third option is to use a paper keyboard (usually referred to as a menu) and to enter in the Z-values as a digitized point. Figure 11-3 is a menu for entering Z-values. The source map and the paper keyboard (or menu) are both placed on the digitizer tables. A program uses two reference points to locate the menu and determine the coordinates of each square on the menu.

1	2	3	4
5	6	7	8
9	0	PUNCH	PRINT
ERASE LAST X,Y COORD. ENTERED	ERASE LAST Z-VALUE ENTERED	END DATA SET	.

A typical sequence for digitizing X,Y,Z data sets might be as follows:

1. Digitize menu reference coordinates C1 and C2.

2. Digitize the appropriate switch (by placing digitizer cursor in approximate center of square) to PUNCH and/or PRINT data.

3. Digitize X,Y coordinate pair from map or graph etc.

4. Enter the corresponding Z-value by digitizing the appropriate number keys (squares) and decimal point.

5. Repeat steps 3 and 4 as necessary.

6. When all X,Y and Z values for one data set are entered digitize the END DATA SET key. Another data set may be digitized by starting at step 1.

7. Note that corrections can be made by digitizing the ERASE keys immediately after making an error.

Fig. 11-3 Digitizer Menu

To digitize a Z-value following the digitizing of an X,Y coordinate, the pen is placed anywhere in the square for each numeral (and decimal point) required to make up the Z-value.

Manipulation of Spatial Data

Although a digitizer removes most of the drudgery of obtaining Cartesian coordinates for geographical locations on a map or diagram, the data usually must be further processed (or preprocessed) before using it in a mapping routine. Appendix E contains the listing for a subroutine (DMANIP) to read in and manipulate spatial data. The instructions for the program are shown on the next page, followed by an explanation of the options allowed.

Instructions for Subroutine DMANIP

The subprogram reads 2 or 3 control cards and then the XYZ data cards.

CARD 1:	TITLE CARD
COLS1-80	ANY TITLE
CARD 2:	OPTIONS CARD (1=YES, 0+NO ON OPTIONS WITH ?)
COL 1	STANDARD FORMAT? (SEE EXAMPLE)
2	INVERT X AXIS?
3	INVERT Y AXIS?
4	INTERCHANGE X AND Y AXIS
5	MANIPULATE X VALUES[1]
6-15	X CONSTANT[2]
16	MANIPULATE Y VALUES[1]
17-26	Y CONSTANT[2]
27	MANIPULATE Z VALUES[1]
28-37	ZCONSTANT[2]
38	ZERO THE X,Y COORDINATES?
39	PUNCH A NEW DATA SET?
40	ECHO CHECK THE DATA SET?
41-50	XMIN OF SOURCE MAP[2]
51-60	XMAX OF SOURCE MAP[2]
61-70	XMIN OF SOURCE MAP[2]
71-80	YMAX OF SOURCE MAP[2]

[1] MANIPULATE COMMANDS
0=SKIP
1=+CONSTANT
2=-CONSTANT
3=* CONSTANT
4=/ CONSTANT

[2] F10.3 FORMAT

CARD 3:	VARIABLE FORMAT CARD (USE ONLY IF COL 1 OF CARD 2 = 0)
COLS 1-80	ENTER THE FORMAT FOR READING IN X,Y AND Z VALUES ONE OBSERVATION SET PER CARD. START IN COL 1 WITH A LEFT PARENS AND THEN CLOSE WITH A RIGHT PARENS. THE FORMAT MUST INCLUDE AN L1 FIELD TO READ IN A LOGICAL FLAG ON THE LAST DATA CARD.
EXAMPLE:	(3(F10.2,5X),34X,L1)
	THIS FORMAT WILL READ X,Y,Z, IN COLS 1-10,

16-25,31-40 AND A LOGICAL FLAG IN COL 80.
A T (FOR TRUE) MUST BE PUNCHED IN COL 80
OF THE LAST DATA CARD.

IF THE VARIABLE FORMAT OPTION IS NOT USED
THE STANDARD FORMAT IS (3F10.3,49X,L1)

Options in DMANIP

1) Axis inversion: The digitizer has a lower left origin for the X and Y axis. Many line printed mapping programs assume an upper left origin to conform to the mechanical aspects of line printing.

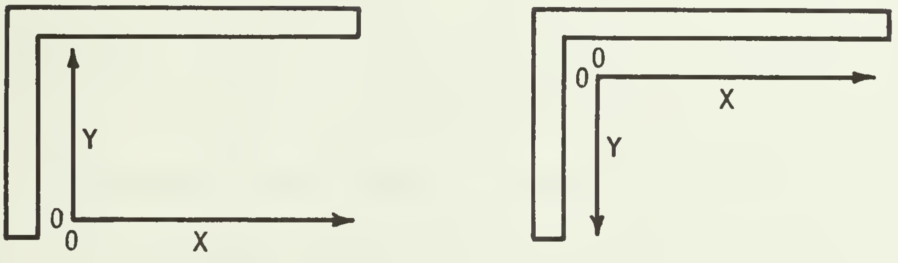


Fig. 11-4 Digitizer Origins

To obtain an upper left origin, the Y axis must be inverted. This can be done by redefining each Y-value as follows.

$$Ynew = Yold -(YMAX + YMIN)$$

The X values may be inverted by the same method. By selection of either or both X and Y axis inversion, the origin can be placed at any of the four corners of the Cartesian quadrant.

2) Interchange X and Y axis: Some data sets may also be defined with axes not conforming to the standard Cartesian configuration. This switch will cause the interchange of the X and Y values.

3) Manipulate X,Y or Z values: Any of these switches allows the addition, subtraction, division or multiplication of a constant to be accomplished with any of the X,Y or Z values. This is particularly useful if two adjacent data sets need to be combined but each was acquired on separate Cartesian coordinate systems. A constant X and or Y value could be added or subtracted from either data set to make it conform to the other (assuming the X axes were parallel to each other; i.e., no rotation is required).

4) Zero the X,Y coordinates: Measurements obtained from a map may be more desirable if referenced to a local origin or map origin rather than to the digitizer origin. This switch will cause the X and Y values to be zeroed to the data minima (XMIN, YMIN). Often the first point digitized on the map may be the desired map origin in the lower left corner. If this is the case and this switch is used, the X and Y values will be referenced to the first point digitized which will then become 0.0,0.0 for the X and Y values. The extremes (XMIN, XMAX, YMIN,YMAX) for a source map can be obtained by digitizing the lower left corner of the map and the upper right corner of the map.

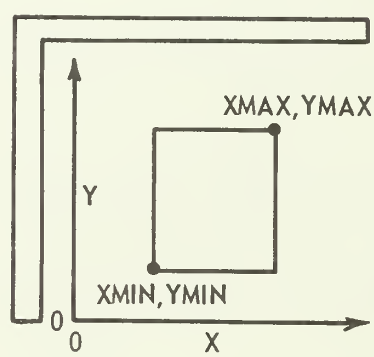


Fig. 11-5 Source Map Extremes

Exercise

Write and implement an algorithm for determining the perimeter and area of a closed figure using digitized coordinates to define the figure. A discussion of the mathematics involved follows this paragraph and should be used as a guide to developing the program.

Perimeter

The perimeter of a closed area is the sum of the straight line segments bounding the area (see Figure 11-6); each segment is calculated from the Pythagorean theorem.

Example: length of the first segment (S1) is obtained as follows:

$$S1 = \quad (X_A - X_B)^2 + (Y_A - Y_B)^2$$

The sum of the straight line segments (perimeter) for a closed area defined by n points is given by:

$$\text{Perimeter} = \sum_{j=1}^{n} [(X_j - X_{j+1})^2 + (Y_j - Y_{j+1})^2]^{\frac{1}{2}}$$

Where: $X_{n+1} = X_1$

$Y_{n+1} = Y_1$

The last (n+1) point is a duplication of the first point in order to close the area. The perimeter of figure 11-4 is:

$$P = [(1-2)^2 + (4-5)^2]^{\frac{1}{2}} + [(2-6)^2 + (5-4)^2]^{\frac{1}{2}} +$$

$$[(6-4)^2 + (4-1)^2]^{\frac{1}{2}} + [(4-3)^2 + (1-2)^2]^{\frac{1}{2}} +$$

$$[(3-2)^2 + (2-2)^2]^{\frac{1}{2}} + [(2-1)^2 + (2-4)^2]^{\frac{1}{2}}$$

$$= 2^{\frac{1}{2}} + 17^{\frac{1}{2}} + 13^{\frac{1}{2}} + 2^{\frac{1}{2}} + 1^{\frac{1}{2}} + 5^{\frac{1}{2}}$$

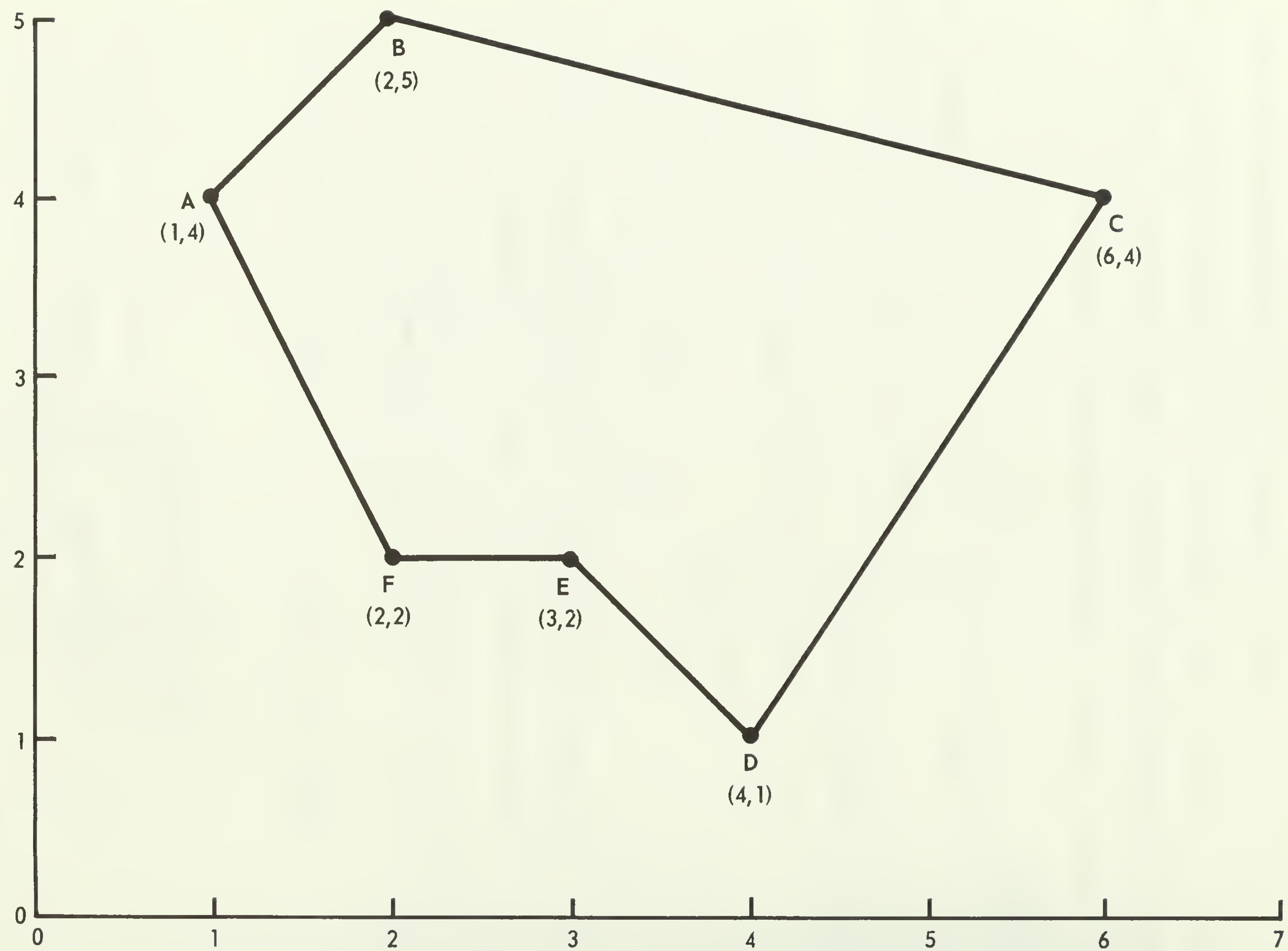


Fig. 11-6 Digitized Sample Area

$$= 1.4 + 4.1 + 3.6 + 1.0 + 2.2$$

$$= 12.3 \text{ units}$$

Area

The area of a closed figure can be found by application of the Coordinate Rule: The area of a closed figure is equal to one-half the sum of the products obtained by multiplying each X-coordinate by the difference between the adjacent X-coordinate taken in the same order around the figure.

Area of Figure 11-6 by the Coordinate Rule

$$A = \tfrac{1}{2} \quad X_A(Y_F - Y_B) + X_B(Y_A - Y_C) + X_C(Y_B - Y_D) + X_D(Y_C - Y_E) + X_E(Y_D - Y_F) + X_F(Y_E - Y_A)$$

$$= \tfrac{1}{2}\,[1(2-5) + 2(4-4) + 6(5-1) + 4(4-2) + 3(1-2) + 2(2-4)]$$

$$= \tfrac{1}{2}\,[-3 + 0 + 24 + 8 - 3 - 4]$$

$$= \tfrac{1}{2}\,[22]$$

$$= 11 \text{ sq. units}$$

Area of Figure 11-6 by Matrix Solution

Point	Y	X	Positive (↘) Cross Multiplication	Negative (↗) Cross Multiplication
A	4	1		
B	5	2	4 x 2 = 8	1 x 5 = 5
C	4	6	5 x 6 = 30	2 x 4 = 8
D	1	4	4 x 4 = 16	6 x 1 = 6
E	2	3	1 x 3 = 3	4 x 2 = 8
F	2	2	2 x 2 = 4	3 x 2 = 6
A	4	1	2 x 1 = 2	2 x 4 = 8
			63	41

$$\text{Area} = \frac{63 - 41}{2} = \frac{22}{2} = 11 \text{ sq. units}$$

LESSON 12: INTRODUCTION TO LINE PLOTTING

Focus:	CALCOMP Programming.
New Statements:	CALL PLOT, CALL FACTOR, CALL SYMBOL
Exercise:	Simple outline map.

Introduction

The CALCOMP output device is capable of drawing pen and ink diagrams, graphs, etc., to a resolution of .001" per pen move. Although there is a large selection of pen movements and routines from which to choose for drawing, it is only necessary to understand a few to accomplish most drawings.

The introductory lesson in CALCOMP programming is focussed on the simple problem of drawing an outline map. This will introduce the programmer to the coordinate system of the CALCOMP, to the basic pen movement for plotting, and to the programming approach to the design of an algorithm for plotting an outline map.

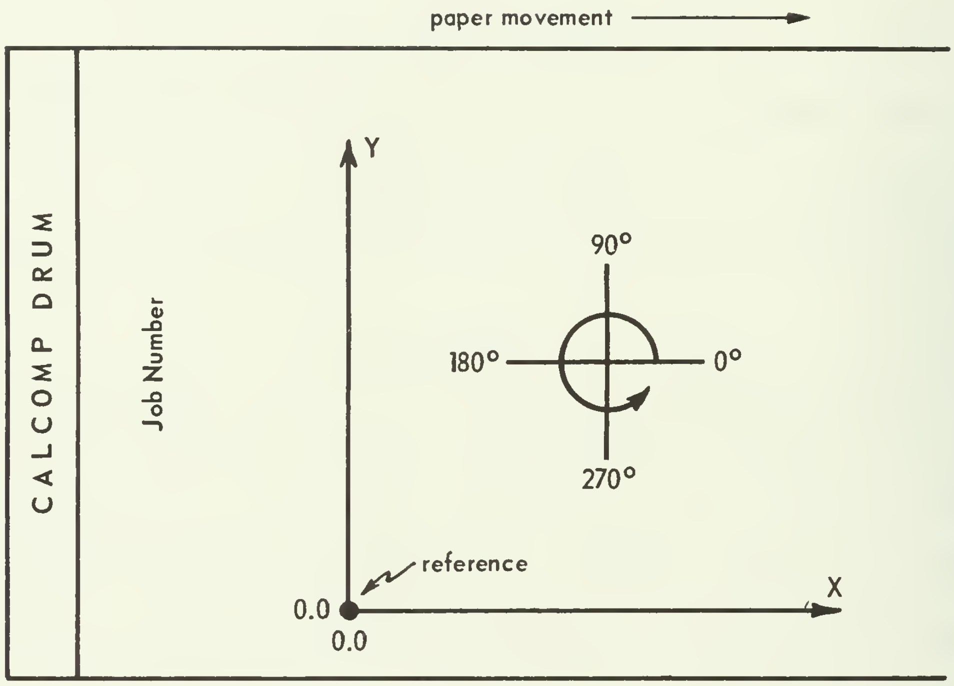


Fig. 12-1 CALCOMP Coordinate System

REF = Current reference point (initialized at beginning of plot)

Y Dimension Maximums: 11'' paper YMAX = 10 inches

30'' paper YMAX = 29 inches

Angles are measured counter-clockwise from X axis.

<u>Basic Pen Movements</u>

CALL PLOT (X,Y,IC)

or CALL CCP1PL (X,Y,IC)

Where: X is the X-coordinate location to which the pen will move

Y is the Y-coordinate location to which the pen will move

IC is the integer control of the pen mode and the (0.0,0.0) reference coordinate

IC = 2 The pen is down (inking) during move to X,Y

IC = -2 The pen is down during move to X,Y and the current reference point (0.0,0.0) is reset to X,Y

IC = 3 The pen is up (not inking) during move to X,Y

IC = -3 The pen is up during move to X,Y and the current reference point (0.0,0.0) is reset to X,Y

The current reference point is initialized at the beginning of the plot but may be reset using a negative IC.

Example:

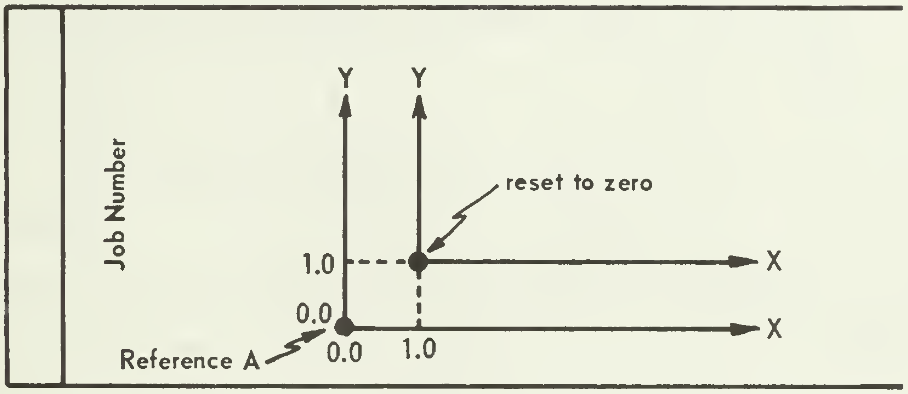


Fig. 12-2 Resetting the Reference Point

A. The system will initialize the reference point for a plot near the bottom of the paper. The Y direction maximum is 10 inches for 11 inch paper.

B. The programmer may reset the reference point with the following statement: CALL PLOT (1.0,1.0,-3). This causes the pen to move (in the <u>up</u> position) to 1.0, 1.0 and then reset to zero. The Y direction maximum is now 9.0 inches.

Using the Full Coordinate System

The coordinate system shown, up to this point, is the positive quadrant of the Cartesian coordinate system. It is possible to use the entire system. For example, if we reset the reference (Figure 12-3) to 5.0,5.0 we could cause the pen to move in the other 3 quadrants.

CALL PLOT (5.0,5.0,-3)

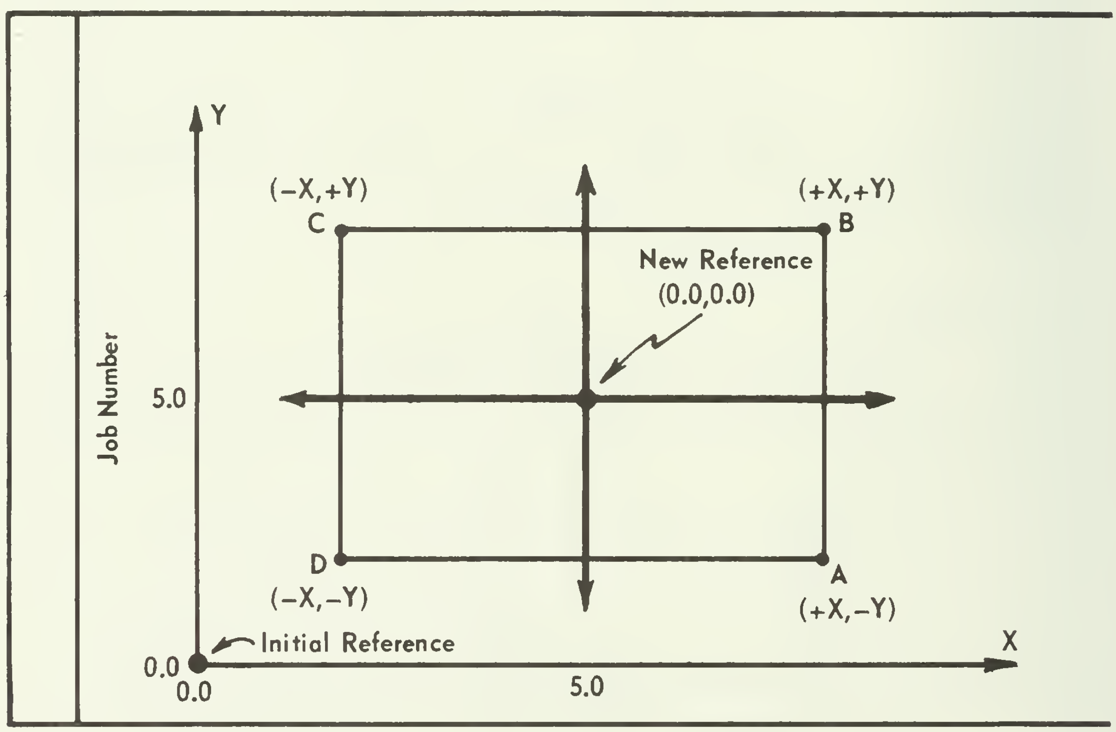


Fig. 12-3 Drawing a Square About the 0.0 Reference

The X and Y coordinates indicated in parentheses show the signs of the coordinates in each quadrant.

Example: To draw a 5" square centered on the new reference, the following calls to plot would be made:

1. To move from the new reference to point A,
   CALL PLOT (2.5,-2.5,3)
2. To move from point A to point B, CALL PLOT (2.5,2.5,2)
3. To move from point B to point C, CALL PLOT (-2.5,2.5,2)

4. To move from point C to point D, CALL PLOT (-2.5,-2.5,2)

5. To move from point D to point A, CALL PLOT (2.5,-2.5,2)

Most of the time it is more convenient to draw within the positive quadrant. Drawing the same 5 inch square from the previous example in the positive quadrant could be done as follows:

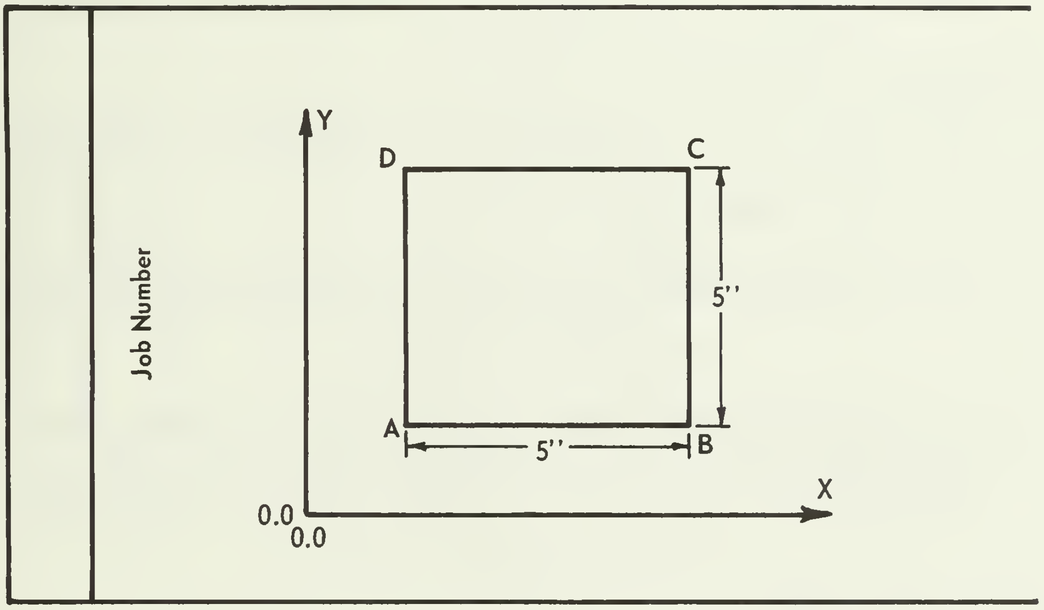


Fig. 12-4 Drawing a Square in the Positive Quadrant

1. To move from reference point to point A and reset reference point, CALL PLOT (2.5,2.5,-3)
2. To move from point A to point B, CALL PLOT (5.0,0.0,2)
3. To move from point B to point C, CALL PLOT (5.0,5.0,2)
4. To move from point C to point D, CALL PLOT (0.0,5.0,2)
5. To move from point D to point A, CALL PLOT (0.0,0.0,2)

Additional Subroutines

Subroutine FACTOR

FACTOR is used to expand or contract all pen moves with respect to the current reference point. All those pen moves made after a call to FACTOR will be altered by FACTOR.

The CALL statement takes the form: CALL CCP1FR (factor)

or

CALL FACTOR (factor)

Where <u>factor</u> is the mulipl ier by which the pen moves are to be increased or decreased.

Multiple calls to FACTOR are not cumulative. Some general uses for FACTOR are shown below:

CALL FACTOR (.394) converts coordinate values in centimeters to plot in inches

CALL FACTOR (2.90) plots ten inch job on 29 inch paper

CALL FACTOR (.33) plots 29 inch job on ten inch paper

Subroutine SYMBOL

The SYMBOL subroutine is used to plot a sequence of alphanumeric information and/or special characters.

The calling sequence takes the forms:

CALL SYMBOL (X,Y,SIZE, EBCDIC,THETA,N)

or

CALL CCP2SY (X,Y,SIZE,EBCDIC,THETA,N)

or

CALL CCP2SB (X,Y,SIZE,EBCDIC,THETA,N)

Where: X,Y - are the coordinates of the lower left corner of the first character to be plotted. The coordinates are given with respect to the current reference point.

SIZE - is the height of the characters in inches. The width of each character and spacing is 6/7* SIZE.

EBCDIC - is the string of characters to be plotted.

THETA - is the angle in degrees at which the characters are to be plotted. (See Figure 12-5)

N - is the number of characters to be plotted.

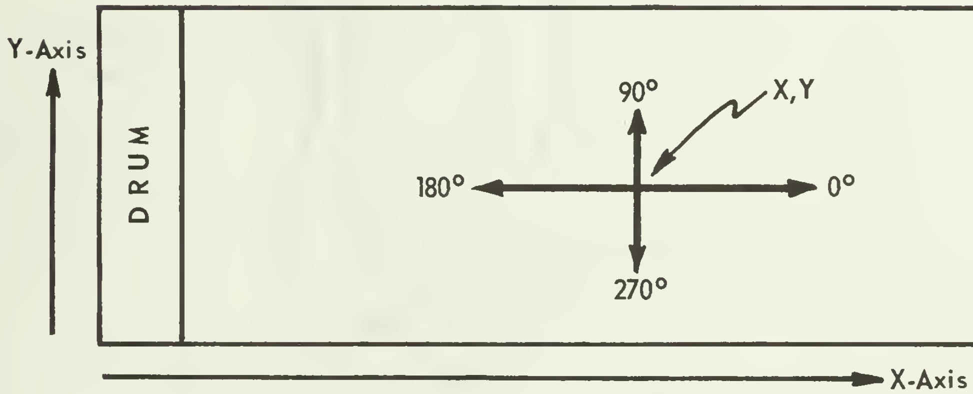


Fig. 12-5 THETA Directions

The ebcdic character string is usually placed in quotes although other alternatives are available (see CALCOMP Manual).

Example:

CALL SYMBOL (0.0,9.5,.25''ILLINOIS'',0.0,8)

will write a title (ILLINOIS) at the top of the paper in $\frac{1}{4}$ inch high letters parallel to the X axis.

Exercise

A Simple Outline Map

To construct a simple outline map from data as shown below.

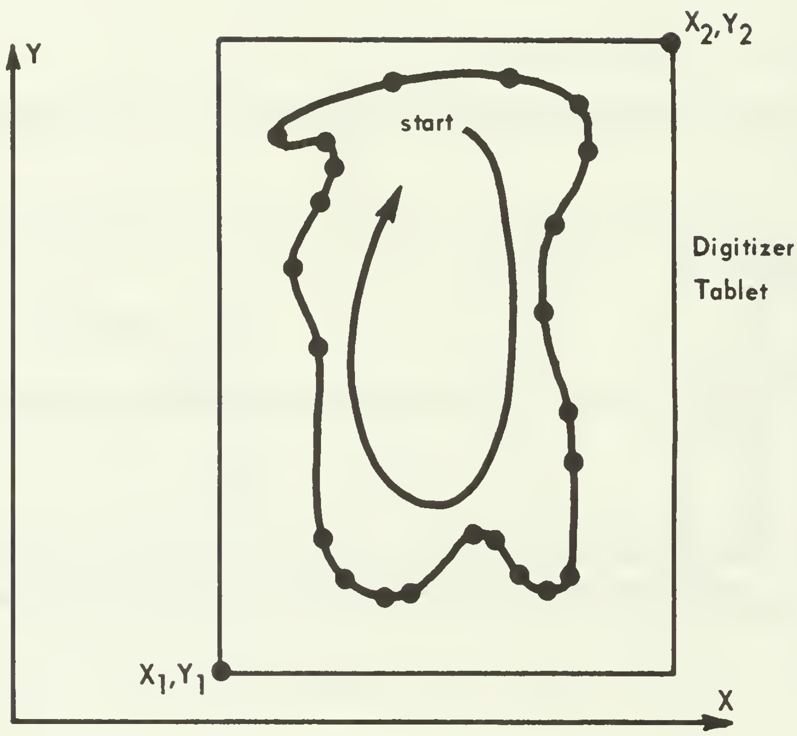


Fig. 12-6 Digitized Outline

1. Digitize minimum (X_1,Y_1) and maximum (X_2,Y_2) of rectangular border confining the outline map.

2. Digitize X,Y coordinates of outline points.

3. Read X_1,Y_1 and X_2,Y_2 and the number of outline points.

```
      READ(5,11) X1,Y1,X2,Y2,NOBS
   11 FORMAT (4F10.2,I3)
```

4. Read X,Y outline points and zero them to X_1,Y_1.

```
      DO 20 J=1,NOBS
      READ(5,21) X(J),Y(J)
   21 FORMAT(2F10.2)
      X(J)=X(J) - X1
   20 Y(J)=Y(J) - Y1
```

5. Calculate factor to fit map on 10'' CALCOMP paper

```
      YDIST=Y2-Y1
```

To allow for titles use a maximum map plotting distance of 9 inches.

```
      FACT=9.0/YDIST
      CALL FACTOR (FACT)
```

6. Duplicate last coordinate so the outline will be closed.

```
      N=NOBS+1
      X(N)=X(1)
      Y(N)=Y(1)
```

7. Plot outline.

   a. Move to first point with pen up

```
      CALL PLOT (X(1),Y(1),3)
```

   b. Loop to plot outline with pen down.

```
      DO 30 J=2,N
   30 CALL PLOT (X(J),Y(J),2)
```

8. Restore original scale.

```
      FACT=YDIST/9.0
      CALL FACTOR (FACT)
```

9. Title the outline

```
      CALL SYMBOL (0.0,9.5,.25,''ILLINOIS,'' 0.0,8)
```

The complete main program is shown below, and the resulting plot is shown in Figure 12-7.

```
      DIMENSION X(200),Y(200)
      READ(5,11)X1,Y1,X2,Y2,NOBS
   11 FORMAT(4F10.2,13)
      DO 20 J=1,NOBS
      READ(5,21)X(J),Y(J)
   21 FORMAT (2.F10.2)
      X(J) = X(J) - X1
   20 Y(J) = Y(J) - Y1
      YDIST=Y2-Y1
      FACT=9.0/YDIST
      CALL FACTOR(FACT)
      N=NOBS+1
      X(N)=X(1)
      Y(N)=Y(1)
      CALL PLOT (X(1),Y(1),3)
      DO 30 J=2,N
   30 CALL PLOT(X(J),Y(J),2)
      FACT=YDIST/9.0
      CALL FACTOR(FACT)
      CALL SYMBOL(0.0,9.5,.25,"ILLINOIS",0.0,8)
      STOP
      END
```

As part of the exercise add the necessary statements to the above program to plot the location (use a # sign) and label a few of the larger cities in Illinois.

Fig. 12-7 Illinois Outline Plot

APPENDIX A

STATISTICAL FORMULAE AND TESTS

Symbols Used

Variable names:	X,Y etc.
Number of observations in a sample:	n
Number of samples:	k
Total number of observation in all samples	N
Population mean:	μ
Sample mean:	$\overline{X}$
Population variance:	σ^2
Population standard deviation:	σ
Sample variance:	s^2
Sample standard deviation:	s
Standard error the mean:	$\hat{\sigma}_{\overline{x}}$
Pooled estimate of population standard deviation:	s_p
Null Hypothesis	H_o
Alternate hypothesis:	H_1

Basic Statistics

Mean ($\overline{X}$)

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

Standard Deviation (s)

$$s = \sqrt{\frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{n-1}}$$

or

$$s = \sqrt{\frac{\sum_{i=1}^{n} X_i^2}{n} - (\overline{X})} \quad \text{(computational formula)}$$

s^2 = variance of the sample

Z-Score

$$\text{Z-Score} = \frac{X - \overline{X}}{s}$$

Standard Error of the Mean ($\hat{\sigma}_{\overline{X}}$)

$$\hat{\sigma}_{\overline{X}} = \frac{s}{\sqrt{n}}$$

Single Sample Test

$$Z \text{ or } t = \frac{\bar{X} - \mu}{\hat{\sigma}_{\bar{x}}}$$

Decision: If calculated Z or t exceeds table value of Z or t the null hypothesis is rejected (see Hypotheses).

Standard Error of the Difference Between Means ($\hat{\sigma}d$)

Unpooled estimate: Use when assumption of similar population variances cannot be met.

$$\hat{\sigma}d = \sqrt{\hat{\sigma}_{\bar{x}1}^2 + \hat{\sigma}_{\bar{x}2}^2}$$

$$= \sqrt{\left[\frac{s_1}{\sqrt{n_1}}\right]^2 + \left[\frac{s_2}{\sqrt{n_2}}\right]^2}$$

Pooled estimate: Use when assumption of similar population variances can be met.

$$\hat{\sigma}d = \sqrt{\hat{\sigma}_{\bar{x}1}^2 + \hat{\sigma}_{\bar{x}2}^2}$$

$$= \sqrt{\left[\frac{s_p}{\sqrt{n_1}}\right]^2 + \left[\frac{s_p}{\sqrt{n_2}}\right]^2}$$

where

$$s_p = \sqrt{\frac{n_1 s_1 + n_2 s_2^2}{n_1 + n_2 - 2}}$$

Means Difference Test

$$Z \text{ or } t = \frac{\overline{X}_1 - \overline{X}_2}{\hat{\sigma}d}$$

Decision: If calculated Z or t exceeds table value of Z or t the null hypothesis is rejected (see Hypotheses).

Hypotheses

Hypothesis	Single Sample Test	Means Difference Test
Null	Ho: $\overline{X} = \mu$	Ho: $\overline{X}_1 = \overline{X}_2$
Alternate (no direction)	H1: $\overline{X} \neq \mu$	H1: $\overline{X}_1 \neq \overline{X}_2$
Alternate (direction - greater than)	H1: $\overline{X} > \mu$	H1: $\overline{X} > \overline{X}_2$
Alternate (direction - less than)	H1: $\overline{X} < \mu$	H1: $\overline{X}_1 < \overline{X}_2$

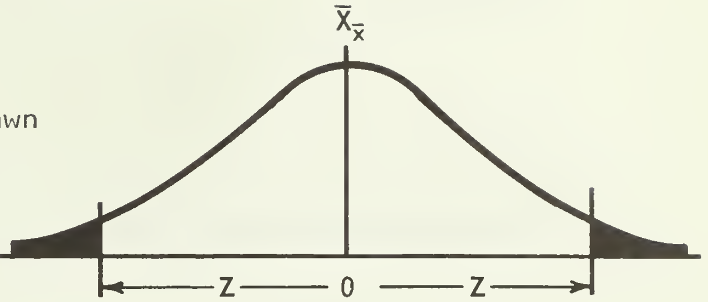
See Table A-1 for significance levels used for large sample testing (Z). Significance levels for small sample testing (t) must be looked up in a t-table.

TABLE A-1

PROBABILITY FOR TESTING SAMPLES

Two Tail Test (No Direction)

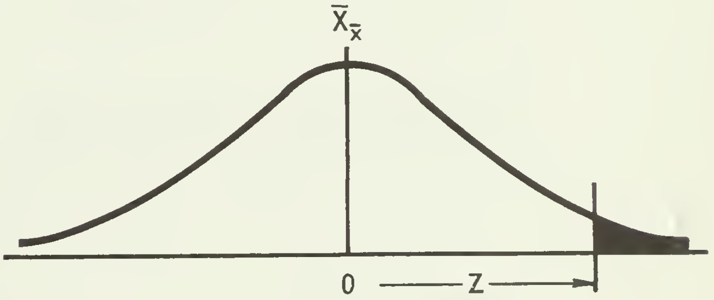
Probability that sample is drawn from population.



Probability	% Observations in Each Tail	± Z Value
1%	.5	2.58
5%	2.5	1.96
10%	5.0	1.64
25%	12.5	1.15
50%	25.0	.67
75%	37.5	.32
100%	50.0	0.00 ($\bar{X}$)

One Tail Test (Direction)

Probability that sample is drawn from population and sample $\bar{X}$ is greater or less than the population $\bar{X}$.



Probability	% Observations in Single Tail	+Z Value or -Z Value
1%	1	2.33
5%	5	1.64
10%	10	1.28
25%	25	.67
50%	50	0.00 ($\bar{X}$)

Analysis of Variance

Hypothesis

$$H_o: \bar{X}_1 = \bar{X}_2 = \bar{X}_3 = \ldots \bar{X}_k$$

$$H_1: \bar{X}_1 \neq \bar{X}_2 \neq \bar{X}_3 \neq \ldots \bar{X}_k$$

Total Sum of Squares (TSS)

$$TSS = \sum_{i=1}^{N} X^2_i - \frac{(\sum_{i=1}^{N} X_i)^2}{N}$$

where

$$N = n_1 + n_2 + n_3 \ldots . n_k$$

k = number of samples

Between Sum of Squares (BSS)

$$BSS = \frac{(\sum_{i=1}^{n_1} X_i)^2}{n_1} + \frac{(\sum_{i=1}^{n_2} X_i)^2}{n_2} + \frac{(\sum_{i=1}^{n_3} X_i)^2}{n_3} \ldots + \frac{(\sum_{i=1}^{n_k} X_i)^2}{n_k} - \frac{(\sum_{i=1}^{N} X_i)^2}{N}$$

Within Sum of Squares (WSS)

$$WSS = TSS - BSS$$

ANOVA Table

Variation	Degrees of Freedom	Estimated Variance
TSS	n-1	
BSS	k-1	BSS/(k-1)
WSS	n-k	WSS/(n-k)

F-statistic

$$F = \frac{\text{Between Estimate of Variance}}{\text{Within Estimate of Variance}}$$

Decision: If calculated F exceeds table F (at k-1 and n-k degrees of freedom) then the null hypothesis is rejected.

Linear Regression and Correlation

Regression Equation

$$Y = a + bX$$

where

Y = dependent variable

X = independent variable

$$a = \frac{(\Sigma Y)(\Sigma X^2) - (\Sigma X)(\Sigma XY)}{N\Sigma X^2 - (\Sigma X)^2} \qquad \text{(Y-intercept)}$$

$$b = \frac{N\Sigma XY - (\Sigma X)(\Sigma Y)}{N\Sigma X^2 - (\Sigma X)^2} \qquad \text{(slope)}$$

Coefficient of Correlation (r)

$$r = \pm\sqrt{\frac{\text{explained variation}}{\text{total variation}}}$$

$$= \pm \frac{N\Sigma XY - (\Sigma X)(\Sigma Y)}{\sqrt{[N\Sigma X^2 - (\Sigma X)^2][N\Sigma Y^2 - (\Sigma Y)^2]}}$$

r^2 = coefficient of determination

Testing r

Ho: $r = 0$

H1: $r \neq 0$

$$F = \frac{r^2 (n-2)}{1 - r^2}$$

Decision: If calculated F exceeds table F (1 and n-2 degrees of freedom) then the null hypothesis is rejected.

APPENDIX B

SAMPLE PROGRAMS USING DO LOOPS

Program to find Minimum and Maximum
of a Vector

```
      DIMENSION X(500)
C     READ IN # OF OBSERVATIONS
      READ(5,21) NOBS
   21 FORMAT(I3)
C     READ IN DATA
      DO 30 J=1, NOBS
   30 READ(5,31) X(J)
   31 FORMAT(F10.2)
C     INITIALIZE MIN AND MAX VALUES
      XMIN= X(1)
      XMAX= X(1)
C     START LOOP FOR FINDING EXTREMES
      DO 40 J=2, NOBS
      XJ=X(J)
      IF(XJ.LT.XMIN)XMIN=XJ
   40 IF(XJ.GT.XMAX)XMAX=XJ
C     WRITE OUT EXTREME VALUES OF VECTOR
      WRITE(6,51) XMAX, XMIN
   51 FORMAT("1",///1X, "MAXIMUM VALUE= ",
     1 F10.2//1X, "MINIMUM VALUE= ",F10.2 ,//"1")
      STOP
      END
```

Program to Sort a Vector in Ascending
or Descending Order

Ascending Order

```
      DIMENSION X(500),XS(500)
C     READ IN # OF OBSERVATIONS
      READ(5,21) NOBS
   21 FORMAT (I3)
C     READ IN DATA
      DO 25 J=1,NOBS
   25 READ(5,26) X(J)
   26 FORMAT(F10.2)
C     START LOOPS FOR SORT
      DO 50 I=1,NOBS
C     INITIALIZE SMALL
      SMALL=X(1)
      DO 40 J=1,NOBS
   45 IF(X(J).LE.SMALL) GO TO 30
      GO TO 40
   30 K=J
      SMALL=X(J)
   40 CONTINUE
      XS(I)=SMALL
   50 X(K)=999999999.
C     WRITE OUT HEADER
      WRITE(6,61) NOBS
   61 FORMAT("1","ARRAY OF ",I3, " OBSERVATIONS",
     1 " SORTED IN ASCENDING ORDER",//)
C     WRITE OUT SORTED ARRAY
      DO 70 I=1,NOBS
   70 WRITE(6,71) XS(I)
   71 FORMAT(1X,F10.2)
C     SKIP TO NEW PAGE
      WRITE(6,72)
   72 FORMAT("1")
      STOP
      END
```

Descending Order

TO MODIFY THE ASCENDING ORDER PROGRAM TO SORT THE ARRAY X INTO THE ARRAY XS IN DESCENDING ORDER, REPLACE THE FOLLOWING STATEMENTS AS FOLLOWS:

```
45 IF(X(J).GE.SMALL) GO TO 30
50 X(K)=-999999999.
61 FORMAT("1","ARRAY OF",I3,"OBSERVATIONS SORTED IN DESCENDING ORDER",//)
```

Descriptive Statistics

GENERAL PROGRAM TO DETERMINE THE DESCRIPTIVE STATISTICS OF A DATA SET. THE STATISTICS INCLUDE THE MEAN, STANDARD DEVIATION, SKEWNESS AND KURTOSIS. ESTIMATES OF THE STANDARD ERROR FOR EACH OF THESE STATISTICS ARE INCLUDED ALONG WITH THE CONFIDENCE LIMITS CALCULATED FROM A USER SPECIFIED SIGNIFICANCE LEVEL. T-TESTS ARE CALCULATED FOR SKEWNESS AND KURTOSIS IN ORDER TO ASSESS THE NORMALITY OF THE DATA.

<u>Instructions</u>

CARD1: TITLE CARD
COLS 1-80 ANY DESIRED TITLE

CARD2: FORMAT AND SIGNIFICANCE LEVEL CARD
COLS 1-72 FORMAT FOR DATA CARDS
EXAMPLE: TO READ ONE OBSERVATION LOCATED IN COLS 21-30 OFF OF EVERY CARD AND TO PLACE AN INTEGER FLAG IN COL 80 OF THE LAST DATA CARD THE FORMAT WOULD LOOK LIKE THIS:
(20X,F10.2,49X,I1)
COLS 73-76 SIGNIFICANCE LEVEL AS A PERCENTAGE I.E., 5 OR 1 MUST BE RIGHT JUSTIFIED
COLS 77-80 SIGNIFICANCE LEVEL FROM T-TABLE I.E., 1.96 OR 2.58, ETC.

CARD3: DATA CARDS WITH ONE OBSERVATION PUNCHED PER CARD ACCORDING TO THE FORMAT SPECIFIED IN CARD2. NOTE THAT THE LAST DATA CARD SHOULD HAVE AN INTEGER (1-9) PUNCHED IN THE I1 FIELD SPECIFIED IN THE FORMAT (CARD2).

Reference: <u>Statistical Methods</u>, G. W. Snedecor, The Iowa State University Press, Ames, Iowa, 1956, pp. 199-202.

```
      DIMENSION X(1000),FORM1(18),ID(20)
C     READ IN TITLE
      READ(5,11)(ID(J),J=1,20)
   11 FORMAT(20A4)
C     READ IN FORMAT FOR DATA AND SIG. LEVEL
      READ(5,21)(FORM1(J),J=1,18),PER,SIG
   21 FORMAT(19A4,F4.2)
C     INITIALIZE COUNTER (I) AND SUMX
      I=0
      SUMX=0.
C     READ IN DATA, COUNT AND SUM
   20 READ(5,FORM1)X(I),IFLAG
      SUMX=SUMX+X(I)
      IF(IFLAG.GT.0) GO TO 30
      I=I+1
      GO TO 20
   30 NOBS=I
      XNOBS=NOBS
```

```
C     LOOP TO SUM FIRST FOUR MOMENTS
      XMEAN=SUMX/XNOBS
      SUMX2=0.
      SUMX3=0.
      SUMX4=0.
      DO 40 J=1,NOBS
      D=X(J)-XMEAN
      SUMX2=SUMX2+D**2
      SUMX3=SUMX3+D**3
   40 SUMX4=SUMX4+D**4
C     CALCULATE REMAINING DESCRIPTIVE STATISTICS
      STDEV=SQRT(SUMX2/XNOBS)
      SKEW=(XNOBS*SUMX3/((XNOBS-1.)*(XNOBS-2.)))/(SQRT((SUMX2/(XNOBS-1.)
     *)**3))
      XKURT=(XNOBS*(((XNOBS+1.)*SUMX4)-(3.*(XNOBS-1.)*SUMX2**2/XNOBS))/(
     *(XNOBS-1.)*(XNOBS-2.)*(XNOBS-3.)))/(SUMX2/(XNOBS-1.))**2
C     CALCULATE STANDARD ERRORS
      XMEANE=STDEV*SQRT(XNOBS/(XNOBS*(XNOBS-1.)))
      STDEVE=STDEV*SQRT(XNOBS/(2.*XNOBS*(XNOBS-1.)))
      SKEWE=SQRT(6.*XNOBS*(XNOBS-1.)/((XNOBS-2.)*(XNOBS+1.)*(XNOBS+3.)))
      XKURTE=SQRT(24.*XNOBS*(XNOBS-1.)**2/((XNOBS-3.)*(XNOBS-2.)*(XNOBS+
     *3.)*(XNOBS+5.)))
C     CALCULATE UPPER CONFIDENCE LIMITS
      XMEANU=XMEAN+SIG*XMEANE
      STDEVU=STDEV+SIG*STDEVE
      SKEWU=SKEW+SIG*SKEWE
      XKURTU=XKURT+SIG*SKEWE
C     CALCULATE LOWER CONFIDENCE LIMITS
      XMEANL=XMEAN-SIG*XMEANE
      STDEVL=STDEV-SIG*STDEVE
      SKEWL=SKEW-SIG*SKEWE
      XKURTL=XKURT-SIG*XKURTE
C     CALCULATE DIMENSIONLESS SKEWNESS AND KURTOSIS
      SKEWD=SKEW/SKEWE
      XKURTD=XKURT/XKURTE
C     WRITE OUT HEADER FOR TABLE
      WRITE(6,61)(ID(J),J=1,20)
   61 FORMAT("1",20A4////9X,"STATISTIC",11X,"VALUE",8X,"STANDARD ERROR",
     *10X,"UCL",13X,"LCL",10X,"Z OR T TEST"/9X,"*********",11X,"*****",8
     *X,"**************",10X,"***",13X,"***",10X,"***********",/)
      WRITE(6,71) XMEAN,XMEANE,XMEANU,XMEANL
   71 FORMAT(9X,"MEAN                ",F10.4,8X,F10.4,8X,F10.4,6X,F10.4,/)
      WRITE(6,72) STDEV,STDEVE,STDEVU,STDEVL
   72 FORMAT(9X,"STD. DEV.           ",F10.4,8X,F10.4,8X,F10.4,6X,F10.4,/)
      WRITE(6,73) SKEW,SKEWE,SKEWU,SKEWL,SKEWD
   73 FORMAT(9X,"SKEWNESS            ",F10.4,8X,F10.4,8X,F10.4,6X,F10.4,7X,
     *F10.4,/)
      WRITE(6,74) XKURT,XKURTE,XKURTU,XKURTL,XKURTD
   74 FORMAT(9X,"KURTOSIS            ",F10.4,8X,F10.4,8X,F10.4,6X,F10.4,7X,
     *F10.4,////)
      WRITE(6,75) NOBS,PER,SIG
   75 FORMAT(9X,"SAMPLE SIZE(N)= ",I4,/9X,"SIGNIFICANCE LEVEL=",A4,"%, "
     *,F4.2," STANDARD DEVIATION UNITS",////)
      RETURN
      END
```

APPENDIX C

MATRICES AND VECTORS

An elementary understanding of the language and manipulation of matrix algebra is important for several reasons:

1. matrix algebra is used to express and understand advanced statistical techniques;
2. geographic data frequently can be arrayed or expressed in matrix form;
3. computer programs rely on matrix techniques.

Here, then, is a summary of a few matrix algebra concepts and the methodology used for manipulation. The following topics will be covered:

1. Matrix Definitions and Notations
2. Algebraic Manipulations
3. Special Matrices
4. Matrix Inversion
5. Solving Simultaneous Equations
6. FORTRAN Matrix Manipulations

Matrix Definitions and Notations

A matrix is a rectangular array of elements. The array, when written, is enclosed by brackets. Letters (A,B) are usually used to identify the matrix as shown below.

$$A = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \qquad B = \begin{bmatrix} 3 & 1 & 2 \\ 2 & 4 & 6 \end{bmatrix}$$

The sets of horizontal elements are called <u>rows</u> and the sets of vertical elements in the array are called <u>columns</u>.

A matrix may be designated by its size or order which is expressed as the number of rows by the number of columns. Matrix A is of size (or order) 2 x 2 and Matrix B is of size 2 x 3. A matrix containing only one row or column is called a <u>row</u> or <u>column vector</u>. Matrix C is a 1 x 3 row vector.

$$C = \begin{bmatrix} 4 & 2 & 3 \end{bmatrix}$$

In the discussion throughout this appendix it should be noted that the 2 x 2 matrix (the smallest, non-vector matrix) is considered a special case. Methods involving 2 x 2 matrices can not usually be applied to larger size matrices in many instances and therefore in most cases methods are given for the 2 x 2 matrix and then for matrices of larger size.

<u>Determinants and Cofactors</u>

The determinant of a matrix is the single scalar value representing a matrix and can be found as follows:

2 x 2 matrix:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\text{DET } A = ad - c$$

3 x 3 matrix:

$$B = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}$$

$$\text{DET } B = a_1b_2c_3 + a_2b_3c_1 + a_3b_1c_2$$
$$a_1b_3c_2 - a_2b_1c_3 - a_3b_2c_1$$

The determinant for the 3 x 3 matrix (and larger) can be found more easily using the cofactor method.

Where

$$B = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}$$

The cofactor of each element of matrix B can be found by "crossing out" the row and column to which the element is common and finding the determinant, then the determinant can easily be found.

$$\text{The cofactor of } a_1 = \text{DET} \begin{bmatrix} b_2 & c_2 \\ b_2 & c_3 \end{bmatrix}$$

$$\text{The cofactor of } a_2 = \text{DET} \begin{bmatrix} b_1 & c_1 \\ b_3 & c_3 \end{bmatrix}$$

Each element of a matrix has a cofactor.

The determinant of B can be found as follows

$$\text{DET } B = a_1 \ (\text{cof. } a_1) - b_1 \ (\text{cof. } b_1) + c_1 \ (\text{cof. } c_1)$$

or

$$a_1 \ \text{DET} \begin{bmatrix} b_2 & c_2 \\ b_3 & c_3 \end{bmatrix} - b_1 \ \text{DET} \begin{bmatrix} a_2 & c_2 \\ a_3 & c_3 \end{bmatrix} + c_1 \ \text{DET} \begin{bmatrix} a_2 & c_2 \\ a_3 & b_3 \end{bmatrix}$$

Example: find the determinant of matrix Y

$$Y = \begin{bmatrix} 2 & -3 & -4 \\ 0 & -4 & 2 \\ 1 & -1 & 5 \end{bmatrix}$$

$$\text{DET } Y = (2)\text{DET}\begin{bmatrix} -4 & 2 \\ -1 & 5 \end{bmatrix} - (3)\text{DET}\begin{bmatrix} 0 & 2 \\ 1 & 5 \end{bmatrix} + (-4)\text{DET}\begin{bmatrix} 0 & -4 \\ 1 & -1 \end{bmatrix}$$

$$= (2)\ (-18) - (3)\ (-2) + (-4)\ (4)$$

$$= -36 + 6 - 16$$

$$= -46$$

Matrix Equality

Two matrices are equal if they are the same size and all corresponding elements are equal.

$$A = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \qquad C = \begin{bmatrix} 2 & 1 \\ 3 & 3 \end{bmatrix}$$

$$A = B \neq C$$

Transposition

Matrices may be transposed by replacing each row by its corresponding column; such a transportation is designated by a prime sign or T.

$$A = \begin{bmatrix} 3 & 6 \\ 4 & 7 \\ 5 & 8 \end{bmatrix} \qquad A^T = \begin{bmatrix} 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

Note that A is a 3 x 2 matrix and the transpose of A (A^T) is a 2 x 3 matrix. Column and row vectors may be transposed in the same way.

Addition and Subtraction

Two matrices may be added together or subtracted from each other only if they are of the same order and size. Addition and subtraction are accomplished by either adding or subtracting elements of one matrix from the corresponding elements of another. Consider two matrices A and B, below: their sums and differences are shown as two new matrices X and Y.

$$A = \begin{bmatrix} -1 & 2 \\ 3 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & -6 \\ 5 & 2 \end{bmatrix}$$

$$X = A + B$$

$$X = \begin{bmatrix} (-1) + 0 & 2 + (-6) \\ 3 + 5 & 4 + 2 \end{bmatrix} = \begin{bmatrix} -1 & -4 \\ 8 & 6 \end{bmatrix}$$

$$Y = A - B$$

$$Y = \begin{bmatrix} (-1) - 0 & 2 - (-6) \\ 3 - 5 & 4 - 2 \end{bmatrix} = \begin{bmatrix} -1 & 8 \\ -2 & 2 \end{bmatrix}$$

It should be noted for addition and subtraction of the transpose of matrices the following is true:

$$A^T + B^T = (A + B)^T$$

$$A^T + B^T = (A - B)^T$$

Multiplication of Matrices

When the number of columns of matrix A is the same as the number of rows for matrix B, A is said to be conformable to B for multiplication or the multiplication is said to be defined for A x B (usually written AB). To see if matrices are conformable for multiplication, place their order values side by side, and if the two inside terms are the same, they are conformable for multiplication and the answer matrix will be of the order of the two outside terms.

Example: $(4,5) \times (5,3) = (4,3)$ $(3,2) \times (1,5) = (3,5)$

Multiplication of matrices is <u>NOT</u> cumulative.

AB does not always equal BA.

Multiplication of matrices <u>IS</u> associative.

$(AB)C = A(BC)$

The following three fundamental properties from scalar algebra <u>DO</u> <u>NOT</u> carry over to matrix algebra:

1. $AB = BA$
2. $AB = 0$
3. $AB = AC$ so $B = C$

To multiply matrices:

1. check for conformability
2. multiple each row of A by each column of B
    a. the sum of the products form (first row of A) x (first column of B) will be the element of the answer matrix in the first row and first column.

b. sum of products (second row A) x (first column of B) will be the element in the answer matrix located in the second row, first column

c. The above procedure is continued until all combinations are completed.

Example:

$$A = \begin{bmatrix} 1 & 4 \\ -3 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} -5 & 1 & 2 \\ 5 & 2 & 1 \end{bmatrix}$$

$$(2 \times 2) \qquad\qquad (2 \times 3)$$

1. Conformability

answer matrix
will be a 2 x 3

$(2 \times 2) \quad (2 \times 3)$

conformable for
multiplication

2. X = AB

$$= \begin{bmatrix} (1)(-5) + (4)(6) & (1)(1) + (4)(2) & (1)(2) + (4)(1) \\ (-3)(-5) + (2)(6) & (-3)(1) + (2)(2) & (-3)(2) + (2)(1) \end{bmatrix}$$

$$= \begin{bmatrix} -5 + 24 & 1 + 8 & 2 + 4 \\ 15 + 12 & -3 + 4 & -6 + 2 \end{bmatrix}$$

$$= \begin{bmatrix} 19 & 9 & 6 \\ 27 & 1 & -4 \end{bmatrix}$$

Similarly, row vectors and column vectors can be multiplied if they are of the same order, and matrices can be multiplied by column or row vectors if they are conformable.

Special Matrices

Square Matrix

A matrix containing the same number of rows and columns is a square matrix. In Matrix A below, the elements 2 and 4 are called the diagonal elements. All other elements are called off-diagonal elements.

$$A = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}$$

Identity Matrix

If in a square matrix all diagonal elements are equal to one and all off-diagonal elements are equal to zero, the matrix is called an identity matrix and is designated by an I.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{or} \quad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

When an identity matrix is involved in multiplication, the results are the same as if the identity matrix were replaced by the scalar 1.

$$IA = A \text{ or } AI = A$$

Zero Matrix

A matrix containing all elements equal to zero is called a zero matrix and behaves as if it were scalar zero.

Diagonal Matrix

A diagonal matrix is a square matrix containing all off-diagonal elements of zero. A is a diagonal matrix.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

Symmetric Matrix

If matrix A is equal to the transpose of A (A^T), then the matrix is said to be symmetric. The matrix A below is equal to A^T,

$$A = A^T = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 9 & 0 \\ 3 & 0 & 8 \end{bmatrix}$$

Matrix Inversion

A matrix A may be inverted (A^{-1}) so that $AA^{-1} = 1$. Not all matrices can be inverted. The matrix has an inverse if the determinant does not equal zero.

A. 2 x 2 Matrix Inversion

(1) find the determinant of the matrix

(2) interchange the elements of the diagonal

(3) multiply the other two elements by -1

(4) divide each element by the determinant

Example: Find the inverse of A.

$$A = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$$

1. $\text{DET } A = (2 \times 5) - (4 \times 3)$

$= 10 - 12$

$= -2$

$\text{DET} \neq 0$ (matrix can be inverted)

2. Interchange the diagonal elements:

$$\begin{bmatrix} 5 & 3 \\ 4 & 2 \end{bmatrix}$$

3. Multiply off diagonal elements by -1:

$$\begin{bmatrix} 5 & -3 \\ -4 & 2 \end{bmatrix}$$

4. Divide each element by the determinant:

$$\begin{bmatrix} 5/-2 & -3/-2 \\ -4/-2 & 2/-2 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} -5/2 & 3/2 \\ 2 & -1 \end{bmatrix}$$

B. 3 x 3 Matrix

(1) find the determinant of the matrix

(2) find the adjoint of the matrix where the adjoint is defined as follows:

$$\text{ADJ } A = \begin{bmatrix} a_1 & -(b_1) & c_1 \\ -(a_2) & b_2 & -(c_2) \\ a_3 & -(b_3) & c_3 \end{bmatrix}$$

where a,b,c are the cofactors of the elements of the original matrix. Note the negative multiplication of certain cofactors in the adjoint.

(3) divide each element of the adjoint by determinant A

(4) transpose the result

Example:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 1 & 2 \end{bmatrix}$$

1. DET A = (1)(0) - (2)(-2) + (3)(-1)

   = 0 + 4 - 3

   = 1 DET A $\neq$ 0 (matrix can be inverted)

2. ADJOINT

$$\text{Matrix of Cofactors} = \begin{bmatrix} 0 & -2 & -1 \\ 1 & 7 & -5 \\ 1 & -4 & -3 \end{bmatrix} \qquad \text{Adjoint} = \begin{bmatrix} 0 & 2 & -1 \\ -1 & -7 & 5 \\ 1 & 4 & -3 \end{bmatrix}$$

3. DIVIDE BY DETERMINANT (DET A=1)

$$\begin{bmatrix} 0 & -2 & -1 \\ -1 & -7 & 5 \\ 1 & 4 & -3 \end{bmatrix}$$

4. TRANSPOSE

$$A^{-1} = \begin{bmatrix} 0 & -1 & 1 \\ 2 & -7 & 4 \\ -1 & 5 & -3 \end{bmatrix}$$

Check: $AA^{-1} = I$

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 1 & 2 \end{bmatrix} \begin{bmatrix} 0 & -1 & 1 \\ 2 & -7 & 4 \\ -1 & 5 & -3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Solving Simultaneous Equations

Linear equations may be expressed in matrix notation and solved using matrix techniques. As an example the three equations in three unknowns shown below can be solved using the techniques of matrix algebra.

System of three equations in three unknowns:

$$2X + Y - Z = 5$$

$$X + Y - Z = 3$$

$$X - 2Y - 3Z = 0$$

In matrix notation the equations can be written as follows:

An S matrix of coefficients

$$S = \begin{bmatrix} 2 & 1 & -1 \\ 1 & 1 & -1 \\ 1 & -2 & -3 \end{bmatrix}$$

A B column vector of unknowns

$$B = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

A g column vector of constant

$$g = \begin{bmatrix} 5 \\ 3 \\ 0 \end{bmatrix}$$

The equations can then be written in matrix form as:

$$SB = g$$

To find the unknowns (solve the equations) we multiply both sides of the matrix equation by the inverse of S.

$$S^{-1}SB = S^{-1}g$$

where $S^{-1}S = I$ (the identity matrix)

$$\therefore \; I\,B = S^{-1}g$$

since I acts as scaler one (1)

$$B = S^{-1}g$$

$$S^{-1} = \begin{bmatrix} -1/5 & 1 & 3/5 \\ 4/5 & -1 & -3/5 \\ -3/5 & 1 & -1/5 \end{bmatrix}$$

$$B = S^{-1}g$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} -1/5 & 1 & 2/5 \\ 4/5 & -1 & -3/5 \\ -3/5 & 1 & -1/5 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 5(-1/5) + 3(1) & + 0(2/5) \\ 5(4/5) + 3(1-1) & + 0(-3/5) \\ 5(-3/5) - 3(1) & + 0(-1/5) \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} (-1 & + & 3 & + 0) \\ (\ \ 4 & - & 3 & + 0) \\ (-3 & + & 3 & + 0) \end{bmatrix}$$

$$X = 2$$

$$Y = 1$$

$$Z = 0$$

<u>A Regression Analysis Example</u>

In the case of simple linear regression, the line of best fit is described by the slope Y-intercept form for a straight line ($Y = a + bX$). The constants a and b are derived from the normal equations which were given in Appendix A of this handbook as:

$$an + b\Sigma X = \Sigma Y$$

(normal equations derived from the calculus)

$$a\Sigma Y + b\Sigma X^2 = \Sigma XY$$

and when solved for a and b yield the familiar computational formulae for the regression coefficients

$$a = \frac{(\Sigma Y)(\Sigma X^2) - (\Sigma X)(\Sigma XY)}{n\Sigma X^2 - (\Sigma X)^2}$$

$$b = \frac{n\Sigma XY - (\Sigma X)(\Sigma Y)}{n\Sigma X^2 - (\Sigma X)^2}$$

The normal equations shown above when written in matrix notation are as follows:

S-MATRIX OF COEFFICIENTS:

$$S = \begin{bmatrix} n & \Sigma X \\ \Sigma X & \Sigma X^2 \end{bmatrix}$$

B-VECTOR OF UNKNOWNS:

$$B = \begin{bmatrix} a \\ b \end{bmatrix}$$

g-VECTOR OF CONSTANTS

$$g = \begin{bmatrix} \Sigma Y \\ \Sigma XY \end{bmatrix}$$

The normal equations can then be written as $SB = g$ and the solution of this set of simultaneous equations is $B = S^{-1}g$.

The inverse of S is done in the following steps:

1. Find the determinant of S

   $$\text{DET } S = n\Sigma X^2 - \Sigma X \Sigma X$$

2. Interchange main diagonal elements of S

   $$\text{then } S = \begin{bmatrix} \Sigma X^2 & \Sigma X \\ \Sigma X & n \end{bmatrix}$$

3. Multiply the off diagonal elements by -1

   $$\text{then } S = \begin{bmatrix} \Sigma X^2 & -\Sigma X \\ -\Sigma X & n \end{bmatrix}$$

4. Divide each element by the determinant to obtain inverse of S

$$S^{-1} = \begin{bmatrix} \dfrac{\Sigma X^2}{n\Sigma X^2 - (\Sigma X)^2} & \dfrac{-\Sigma X}{n\Sigma X^2 - (\Sigma X)^2} \\[2ex] \dfrac{-\Sigma X}{n\Sigma X^2 - (\Sigma X)^2} & \dfrac{n}{n\Sigma X^2 - (\Sigma X)^2} \end{bmatrix}$$

The system $B = S^{-1}g$ can now be solved:

$$\begin{array}{ccccc} B & & S^{-1} & & g \\ \begin{bmatrix} a \\[4ex] b \end{bmatrix} & = & \begin{bmatrix} \dfrac{\Sigma X^2}{n\Sigma X^2 - (\Sigma X)^2} & \dfrac{-\Sigma Y}{n\Sigma X^2 - (\Sigma X)^2} \\[2ex] \dfrac{\Sigma X}{n\Sigma X^2 - (\Sigma X)^2} & \dfrac{n}{n\Sigma X^2 - (\Sigma X)^2} \end{bmatrix} & & \begin{bmatrix} \Sigma Y \\[4ex] \Sigma XY \end{bmatrix} \end{array}$$

$$\begin{bmatrix} a \\[4ex] b \end{bmatrix} = \begin{bmatrix} \dfrac{\Sigma X^2 \Sigma Y}{n\Sigma X^2 - (\Sigma X)^2} + \dfrac{-\Sigma X \Sigma XY}{n\Sigma X^2 - (\Sigma X)^2} \\[2ex] \dfrac{-\Sigma X \Sigma Y}{n\Sigma X^2 - (\Sigma X)^2} + \dfrac{n\Sigma XY}{n\Sigma X^2 - (\Sigma X)^2} \end{bmatrix}$$

Then:

$$a = \frac{(\Sigma Y)(\Sigma X^2) - (\Sigma X)(\Sigma XY)}{n\Sigma X^2 - (\Sigma X)^2}$$

$$b = \frac{n\Sigma XY - (\Sigma X)(\Sigma Y)}{n\Sigma X^2 - (\Sigma X)^2}$$

FORTRAN Matrix Manipulations

Transposition

```
      DO 10 I = 1,N
      DO 10 J + 1,M
10    B(J,I)  = A(I,J)
```

Where:

N = number of rows in matrix A
M = number of columns in matrix A
$B = A^T$

Addition

```
      DO 10 I = 1,N
      DO 10 J = 1,M
10    C(I,J)  = A(I,J) + B(I,J)
```

Where:

N = number of rows in matrix A or B
M = number of columns in matrix A or B
C = A + B

Multiplication of Two Matrices

```
      DO 10 I = 1,NA
      DO 10 J = 1,MB
      C(I,J)  = 0.
      DO 10 K = 1,MA
10    C(I,J)  = C(I,J) + A(I,K)*B(K,J)
```

Where:

NA = number of rows in matrix A
MB = number of columns in matrix B
MA = number of columns in matrix A
C = A*B

Vector-Matrix Multiplication

Premultiplication of a matrix by a row vector:

```
      DO 10 J = 1,M
      PV(J)   = 0.
      DO 10 I = 1,N
10    PV(J)   = PV(J)+V(I)*A(I,J)
```

Where:

M = number of columns in matrix A or number of columns in product vector PV

N = number of rows in matrix A or number of columns in vector V

PV = V*A

Postmultiplication of a matrix by a column vector:

```
      DO 10 I = 1,N
      PV(I)   = 0.
      DO 10 J = 1,M
10    PV(I)   = PV(I)+A(I,J)*V(J)
```

Where:

M = number of columns in matrix A or number of rows in vector V

N = number of rows in matrix A or number of rows in product vector PV

PV = A*V

Simultaneous Equations

In the previous section the solution of a set of simultaneous equations was carried out using matrix inversion. A simpler approach is to use a method that eliminates terms. A subroutine (EQUAT) is given below which solves a set of simultaneous equations that are given in matrix form. In order to see the relationships between the set of equations and the matrix form of the equations a simple example is given to illustrate the use of subroutine EQUAT.

Set of simultaneous equations:

$$
\begin{aligned}
2X + Y - Z &= 5 \\
X + Y - Z &= 3 \\
X - 2Y - 3Z &= 0
\end{aligned}
$$

Matrix form of the equations:

$$
\underset{S}{\begin{bmatrix} 2 & 1 & -1 \\ 1 & 1 & -1 \\ 1 & -2 & -3 \end{bmatrix}}
\underset{B}{\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}}
=
\underset{g}{\begin{bmatrix} 5 \\ 3 \\ 0 \end{bmatrix}}
$$

$S \times B = g$

Augmented S Matrix:

The S matrix can be augmented by adding the g vector as the fourth column. The new matrix (SA) is now a 3 x 4 matrix.

$$
SA = \begin{bmatrix} 2 & 1 & -1 & 5 \\ 1 & 1 & -1 & 3 \\ 1 & -2 & -3 & 0 \end{bmatrix}
$$

Solving the Equations:

CALL EQUAT (SA, NROWS, NCOLS)

The equations will be solved by elimination and the answer vector will be found in the fourth column of SA returned from the subroutine.

$$
SA = \begin{bmatrix} 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}
$$

```
      SUBROUTINE EQUAT (SA, NROWS, NCOLS)
      DIMENSION SA (NROWS, NCOLS)
C     SA = AUGMENTED S MATRIX
C     ANSWER VECTOR WILL BE NCOLS COLUMN OF SA
C     NROWS = NUMBER OF ROWS IN SA
C     NCOLS = NUMBER OF COLUMNS IN SA
C
C     SOLVE EQUATIONS BY ELIMINATION
      DO 20 I = 1, NROWS
      II = I + 1
      DO 20 J = II, NCOLS
      SA(I,J) = SA(I,J)/SA(I,I)
      DO 20 K = 1, NROWS
      IF (I-K) 10,20,10
   10 SA(K,J) =  SA(K,J)-SA(K,I)*(I,J)
   20 CONTINUE
      RETURN
      END
```

APPENDIX D

GENERALIZED REGRESSION MODELS

Simple linear regression provides the researcher with a powerful technique by which the relationship between two variables may be assessed. However, in many instances the explanatory power of a simple linear model may be insufficient to deal with complex problems involving several variables. The linear model can be extended to encompass more than one independent variable (multiple regression model) or it can be expanded to include non-linear terms (curvilinear regression model) and a combination of these two models (multiple-curvilinear model) is of particular use to geographers. In actuality the regression models discussed in this appendix are all from the same family of polynomial regression models (see Fig. D-1), but for purposes of discussion they have been broken down into the three categories: multiple, curvilinear and multiple-curvilinear.

This appendix is intended to serve as a guide for the construction of regression models in a format suitable for implementation as a computer algorithm. Three principal topics are covered:

1. Derivation of the normal equations (in matrix notation) by inspection for any regression model.
2. The general form of the normal equations for the curvilinear, multiple and multiple-curvilinear regression models.
3. A summary of variance measures associated with the regression models.

Derivation of Normal Equations by Inspection

The normal equations for regression models are determined through application of the calculus. However, the normal equations can be more readily derived by inspection of the regression model and directly describing the normal equations in matrix form. An example using a 3rd

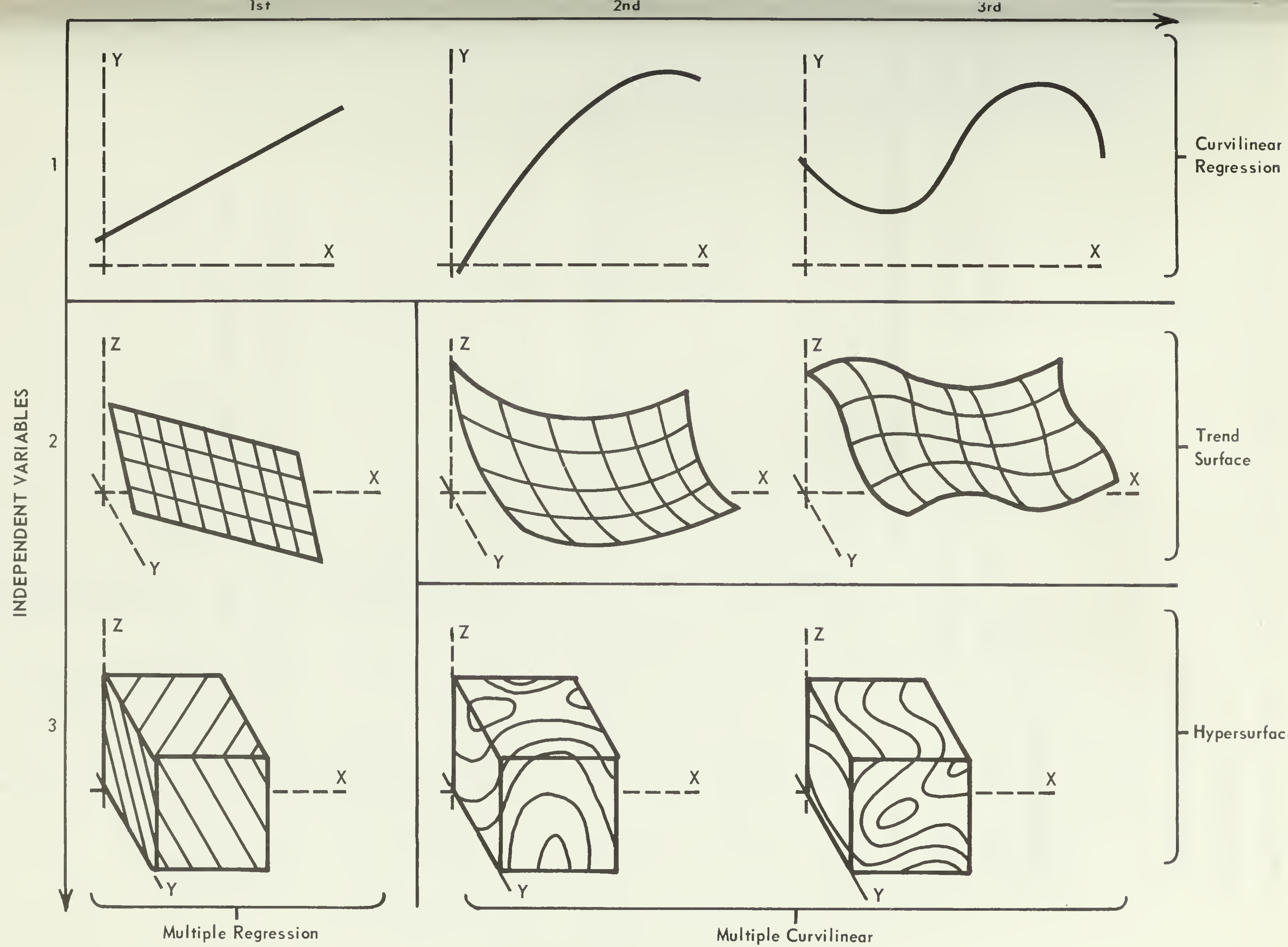


Fig. D-1 Regression Models

degree curvilinear regression model illustrates the procedure applicable to any model.

Regression equation:

$$Y = a_0 + a_1X + a_2X^2 + a_3X^3$$

Generating the S matrix:

Step 1: Using the right side of the regression equation $a_0 + a_1X + a_2X^2 + a_3X^3$ write the variables (X, X^2, X^3) as the first row of the S matrix in columns 2, 3 and 4. The total number of sample observations (n) occupies the first row, first column position.

$$S = \begin{bmatrix} n & X & X^2 & X^3 \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

Step 2: Transpose this row and write the first column of the S-Matrix

$$S = \begin{bmatrix} n & X & X^2 & X^3 \\ X & & & \\ X^2 & & & \\ X^3 & & & \end{bmatrix}$$

Step 3: Fill in the remaining elements by cross multiplication, i.e., for the 2nd row, 2nd column value multiply the 2nd row value of column 1 (X) by the 2nd column row 1, value $(X) = X^2$.

$$S = \begin{bmatrix} n & X & X^2 & X^3 \\ X & X^2 & X^3 & X^4 \\ X^2 & X^3 & X^4 & X^5 \\ X^3 & X^4 & X^5 & X^6 \end{bmatrix}$$

Step 4: Place summation signs $\sum_{j=1}^{n}$ in front of each of the variables and their cross products. For the sake of clarity Σ alone is used assuming j = 1,n.

$$S=\begin{bmatrix} n & \Sigma X & \Sigma X^2 & \Sigma X^3 \\ \Sigma X & \Sigma X^2 & \Sigma X^3 & \Sigma X^4 \\ \Sigma X^2 & \Sigma X^3 & \Sigma X^4 & \Sigma X^5 \\ \Sigma X^3 & \Sigma X^4 & \Sigma X^5 & \Sigma X^6 \end{bmatrix}$$

Generating the B vector of unknowns:

This is simply written as column vector with the regression coefficients written in ascending order down the vector.

$$B=\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Generating the g vector of constants:

Step 1. This is a column vector consisting of the dependent variable (Y) and the cross products of the dependent variable (Y) and the independent variables in the order they appear in the regression equation (X,X^2,X^3).

$$g=\begin{bmatrix} Y \\ YX \\ YX^2 \\ YX^3 \end{bmatrix}$$

Step 2: Place a summation sign $\sum_{j=1}^{n}$ ignoring the counter j = 1,n in front of each vector element

$$g=\begin{bmatrix} \Sigma Y \\ \Sigma YX \\ \Sigma YX^2 \\ \Sigma YX^3 \end{bmatrix}$$

<u>The Normal Equations in Matrix Form</u>

$$\overset{S}{\begin{bmatrix} n & \Sigma X & \Sigma X^2 & \Sigma X^3 \\ \Sigma X & \Sigma X^2 & \Sigma X^3 & \Sigma X^4 \\ \Sigma X^2 & \Sigma X^3 & \Sigma X^4 & \Sigma X^5 \\ \Sigma X^3 & \Sigma X^4 & \Sigma X^5 & \Sigma X^6 \end{bmatrix}} \cdot \overset{B}{\begin{bmatrix} a_o \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}} = \overset{g}{\begin{bmatrix} \Sigma Y \\ \Sigma YX \\ \Sigma YX^2 \\ \Sigma YX^3 \end{bmatrix}}$$

Curvilinear Regression

<u>General form of the regression equation:</u>

$$Y = a_0 + a_1X + a_2X^2 + a_3X^3 + a_4X^4 \; \ldots\ldots\ldots + a_nX^n$$

<u>Description of the normal equations:</u>

$$S=\begin{bmatrix} n & \Sigma X & \Sigma X^2 & \Sigma X^3 & \ldots\ldots & \Sigma X^n \\ \Sigma X & \Sigma X^2 & \Sigma X^3 & \Sigma X^4 & \ldots\ldots & \Sigma X^{n+1} \\ \Sigma X^2 & \Sigma X^3 & \Sigma X^4 & \Sigma X^5 & & \Sigma X^{n+2} \\ \Sigma X^3 & \Sigma X^4 & \Sigma X^5 & \Sigma X^6 & \ldots\ldots & \Sigma X^{n+3} \\ \vdots & \ddots & & & & \\ \Sigma X^n & \Sigma X^{n+1} & \Sigma X^{n+2} & \Sigma X^{n+3} & \ldots\ldots & \Sigma X^{2n} \end{bmatrix}$$

$$B = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix}$$

$$g = \begin{bmatrix} \Sigma Y \\ \Sigma YX \\ \Sigma YX^2 \\ \Sigma YX^3 \\ \vdots \\ \Sigma YX^n \end{bmatrix}$$

Solution of normal equations:

$$B = S^{-1}g$$

Multiple Regression

General form of the regression equations:

$$Y = a_0 + a_1X_1 + a_2X_2 + \ldots\ldots + a_nX_n$$

Description of the normal equations:

$$S = \begin{bmatrix} n & \Sigma X_1 & \Sigma X_2 & \Sigma X_3 & \ldots\ldots & \Sigma X_n \\ \Sigma X_1 & \Sigma X_1^2 & \Sigma X_1X_2 & \Sigma X_1X_3 & \ldots\ldots & \Sigma X_1X_n \\ \Sigma X_2 & \Sigma X_1X_2 & \Sigma X_2^2 & \Sigma X_2X_3 & \ldots\ldots & \Sigma X_2X_n \\ \Sigma X_3 & \Sigma X_1X_3 & \Sigma X_2X_3 & \Sigma X_3^2 & \ldots\ldots & \Sigma X_3X_n \\ \vdots & & & & & \\ \Sigma X_n & \Sigma X_1X_n & \Sigma X_2X_n & \Sigma X_3X_n & \ldots\ldots & \Sigma X_n^2 \end{bmatrix}$$

$$B = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix}$$

$$g = \begin{bmatrix} \Sigma Y \\ \Sigma YX_1 \\ \Sigma YX_2 \\ \Sigma YX_3 \\ \vdots \\ \Sigma YX_n \end{bmatrix}$$

Solution of the normal equations:

$$B = S^{-1}g$$

Multiple-Curvilinear Regression

The more general polynomial regression model is one in which the number of independent variables vary and the degree of the polynomial may be raised. Although there are an indefinite number of such models, geographers have a particular interest in two of them:

1. A Trend Surface model of 1 dependent and two independent variables with the "degree varying."
2. A Hypersurface model of 1 dependent and three independent variables with the "degree varying."

The normal equations for a trend surface analysis are discussed below.

General form of the equations:

1st degree: $Z = a_0 + a_1X + a_2Y$

2nd degree: $Z = a_0 + a_1X + a_2Y + \underline{a_3X^2 + a_4XY + a_5Y^2}$

3rd degree: $Z = a_0 + a_1X + a_2Y + a_3X^2 + a_4XY + a_5Y^2 + a_6X^3 + a_7X^2Y$

$$\underline{+ a_8XY^2 + a_9Y^3}$$

The underlined portion shows the group of terms added to the previous equation to generate the model of the next highest degree. This is unlike the curvilinear or multiple case where the next higher order (higher degree of additional independent variables) is generated by simply adding one extra term. In the multiple-curvilinear case a group of terms is added to generate the next highest order regression equation.

Description of the normal equations:

Because the general case is complex, a description of the normal equations in matrix form for a second degree trend surface is given as an example.

$$S = \begin{bmatrix} n & \Sigma X & \Sigma Y & \Sigma X^2 & \Sigma XY & \Sigma Y^2 \\ \Sigma X & \Sigma X^2 & \Sigma XY & \Sigma X^3 & \Sigma X^2Y & \Sigma XY^2 \\ \Sigma Y & \Sigma XY & \Sigma Y^2 & \Sigma X^2Y & \Sigma XY^2 & \Sigma Y^3 \\ \Sigma X^2 & \Sigma X^3 & \Sigma X^2Y & \Sigma X^4 & \Sigma X^3Y & \Sigma X^2Y^2 \\ \Sigma XY & \Sigma X^2Y & \Sigma XY^2 & \Sigma X^3Y & \Sigma X^2Y^2 & \Sigma XY^3 \\ \Sigma Y^2 & \Sigma XY^2 & \Sigma Y^3 & \Sigma X^2Y^2 & \Sigma XY^3 & \Sigma Y^4 \end{bmatrix}$$

$$B = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}$$

$$g = \begin{bmatrix} \Sigma Z \\ \Sigma ZX \\ \Sigma ZY \\ \Sigma ZX^2 \\ \Sigma ZXY \\ \Sigma ZY^2 \end{bmatrix}$$

Solution of the normal equations:

$$B = S^{-1}g$$

Variation and Correlation Measures

A general approach to the correlation measures for any regression model can be made through an analysis of variance.

Variation and variance can be defined as follows:

Variation or sum of squares (SS) = sums of squares of deviations

$$\text{Variance or mean square (MS)} = \frac{\text{sums of squares of deviations}}{\text{degrees of freedom}}$$

For any regression model the specific measures of variation are obtained as follows:

$\Sigma(Y - \hat{Y})^2$ = unexplained variation (USS)

$\Sigma(Y - \bar{Y})^2$ = explained variation (ESS)

$\Sigma(Y - \bar{Y})^2$ = total variation (TSS)

where:

Y = sample value of the dependent variable

$\hat{Y}$ = estimate of predicted value obtained from the regression equation using sample values of the independent variable(s).

$\bar{Y}$ = the mean of the sample values of the dependent variable

The coefficient of determination (r^2) and coefficient of correlation (r) can then be determined for any regression model from the variation estimates.

Coefficient of determination (r^2)

$$= \frac{\text{explained variation}}{\text{total variation}}$$

$$= \frac{\Sigma(\hat{Y} - \bar{Y})^2}{\Sigma(Y - \bar{Y})^2}$$

Coefficient of correlation (r):

$$= \sqrt{\frac{\text{explained variation}}{\text{total variation}}}$$

$$= \sqrt{\frac{\Sigma(\hat{Y} - \bar{Y})^2}{\Sigma(Y - \bar{Y})^2}}$$

An analysis of variance (F-test) can be applied as a validity check on the coefficient of correlation to determine whether the coefficient of correlation occurred by chance or not. The following ANOVA table can be used for all regression models.

Variation (Sum of Squares)	Degrees of Freedom	Variance (Mean Square)
Total (TSS)	$n - 1$	TMS = TSS/(n-1)
Explained (ESS)	m	EMS = ESS/m
Unexplained (USS)	$n - m - 1$	UMS = USS/(n-m-1)

Where n = number of sets of observations

m = number of terms in regression equation

F-Statistic:

$Ho: r = 0$

$H_1: r \neq 0$

$F = EMS/UMS$

Decision: If F calculated exceeds F table (at m and n-m-1 degrees of freedom) then Ho is rejected and H_1 accepted.

APPENDIX E

LISTING FOR SUBROUTINE DMANIP

```
      SUBROUTINE DMANIP(X,Y,Z,SMX,DSX,NOBS,ID)
      DIMENSION X(1),Y(1),Z(1),SMX(4),DSX(6)
      DIMENSION FORM1(20),ID(20),STX(4)
      LOGICAL*1 LAST
      K=1
C
C     READ TITLE CARD
      READ(5,101)(ID(J),J=1,20)
  101 FORMAT(20A4)
C
C     READ OPTIONS CARD
      READ(5,111) ISF,IX,IY,IXY,MX,XCONST,MY,YCONST,MZ,ZCONST,IZERO,I
     *CH,IECHO,(SMX(J),J=1,4)
  111 FORMAT(5I1,3(F10.3,I1),2I1,4F10.3)
      DO 120 J=1,4
  120 STX(J)=SMX(J)
C
C     CHECK FOR VARIABLE FORMAT CARD
      IF(ISF.EQ.1)GO TO 125
      READ(5,101)(FORM1(J),J=1,20)
      GO TO 150
C
C     READ IN DATA USING STANDARD FORMAT
  125 READ(5,131)X(K),Y(K),Z(K),LAST
  131 FORMAT(3F10.3,49X,L1)
      IF(LAST)GO TO 140
      K=K+1
      GO TO 125
  140 NOBS=K
      GO TO 170
C
C     READ IN DATA USING VARIABLE FORMAT
  150 READ(5,FORM1)X(K),Y(K),Z(K),LAST
      IF(LAST)GO TO 160
      K=K+1
      GO TO 150
  160 NOBS=K
C
C     SUM XMIN+XMAX AND SUM YMIN AND YMAX
  170 SUMX=SMX(1)+SMX(2)
      SUMY=SMX(3)+SMX(4)
C
C     CHECK FOR INVERSION OF X AXIS
      IF(IX.EQ.0)GO TO 190
      DO 180 J=1,NOBS
  180 X(J)=SUMX-X(J)
C
C     CHECK FOR INVERSION OF Y AXIS
```

```
  190 IF(IY.EQ.0)GO TO 210
      DO 200 J=1,NOBS
  200 Y(J)=SUMY-Y(J)
C
C     CHECK FOR INTERCHANGE OF AXES
  210 IF(IXY.EQ.0)GO TO 300
      DO 220 J=1,NOBS
      TEMP1=X(J)
      X(J)=Y(J)
  220 Y(J)=TEMP1
      TEMP1=SMX(1)
      SMX(1)=SMX(3)
      SMX(3)=TEMP1
      TEMP1=SMX(2)
      SMX(2)=SMX(4)
      SMX(4)=TEMP1
C
C     CHECK FOR MANIPULATION OF X-COORDINATES
  300 IF(MX.EQ.0)GO TO 400
      CALL MANIP(X,XCONST,MX,SMX(1),SMX(2),NOBS)
C
C     CHECK FOR MANIPULATION OF Y-COORDINATES
  400 IF(MY.EQ.0)GO TO 500
      CALL MANIP(Y,YCONST,MY,SMX(3),SMX(4),NOBS)
C
C     CHECK FOR MANIPULATION OF Z-COORDINATES
  500 IF(MZ.EQ.0)GO TO 505
      CALL MANIP(Z,ZCONST,MZ,0.0,0.0,NOBS)
C
C     CHECK FOR ZEROING COORDINATES
  505 IF(IZERO.EQ.0)GO TO 600
      DO 510 J=1,NOBS
      Y(J)=Y(J)-SMX(3)
  510 X(J)=X(J)-SMX(1)
      SMX(2)=SMX(2)-SMX(1)
      SMX(1)=0.0
      SMX(4)=SMX(4)-SMX(3)
      SMX(3)=0.0
C
C     SORT THE DATA VECTORS FOR MIN AND MAX VALUES
  600 CALL HILO(X,NOBS,DSX(1),DSX(2))
      CALL HILO(Y,NOBS,DSX(3),DSX(4))
      CALL HILO(Z,NOBS,DSX(5),DSX(6))
C
C     CHECK FOR PUNCHING A NEW DATA DECK
      IF(IPUNCH.EQ.0)GO TO 700
      WRITE(7,601)(ID(J),J=1,20)
  601 FORMAT(20A4)
```

```
      DO 610 J=1,NOBS
  610 WRITE(7,611)J,X(J),Y(J),Z(J)
  611 FORMAT(I4,5X,3F10.3)
C
C     WRITE OUT STATUS OF ALL OPTIONS
  700 WRITE(6,691)(ID(J),J=1,20)
  691 FORMAT("1","*****************",/1X,"*OPTIONS LISTING*",2X,20A4,/1X
     *,"*****************",//4X,"(1=YES,0=NO)",//)
      WRITE(6,692)ISF,IX,IY,IXY,IZERO,IPUNCH,IECHO,MX,XCONST,MY,YCONST,M
     *Z,ZCONST,(STX(J),J=1,4),(SMX(J),J=1,4),(DSX(J),J=1,6),NOBS
  692 FORMAT(1X,"STANDARD FORMAT= ",I1//1X,"INVERT X AXIS= ",I1//1X,"INV
     *ERT Y AXIS= ",I1//1X,"INTERCHANGE X AND Y AXES= ",I1//1X,"ZERO X A
     *ND Y AXES= ",I1//1X,"PUNCH A NEW DATA SET= ",I1//1X,"ECHO THE DATA
     * SET= ",I1//1X,"MANIPULATE THE X,  Y, OR Z VECTORS",    //3X,"1=VE
     *CTOR + CONSTANT",/3X,"2=VECTOR - CONSTANT",/3X,"3=VECTOR * CONSTAN
     *T",/3X,"4=VECTOR / CONSTANT",//3X,"MANIPULATE X-VECTOR= ",I1,/12X,
     *"X-CONSTANT= ",F10.3,//3X,"MANIPULATE Y-VECTOR= ",I1/12X,"Y-CONSTA
     *NT= ",F10.3,//3X,"MANIPULATE Z-VECTOR= ",I1,/12X,"Z-CONSTANT= ",F1
     *0.3,///1X,"ORIGINAL SOURCE MAP EXTREMES",//3X,"XMINSM= ",F10.3,/3X
     *,"XMAXSM= ",F10.3,/3X,"YMINSM= ",F10.3,/3X,"YMAXSM= ",F10.3,///1X,
     *"MODIFIED SOURCE MAP EXTREMES",// 3X,"XMINMS= ",F10.3,/3X,"XMAXMS=
     * ",F10.3,/3X,"YMINMS= ",F10.3,/3X,"YMAXMS= ",F10.3,///1X,
     *"DATA VECTOR EXTREMES",//3X,"XMINDV= ",F10.3,/3X,"XMAXDV= ",F10.3,
     */3X,"YMINDV= ",F10.3,/3X,"YMAXDV= ",F10.3,
     */3X,"ZMINDV= ",F10.3,/3X,"ZMAXDV= ",F10.3,///1X,"NUMBER OF OBSERVA
     *TIONS= ",I4)
C
C     CHECK FOR DATA ECHO
      IF(IECHO.EQ.0)GO TO 800
      WRITE(6,693)(ID(J),J=1,20)
  693 FORMAT("1","*****************",/1X,"*DATA ECHO CHECK*",2X,20A4,/1X
     *,"***************** ",///1X,"OBSERVATION     X-COORDINATE     Y-CO
     *ORDINATE      Z-VALUE",/)
      DO 710 J=1,NOBS
  710 WRITE(6,711)J,X(J),Y(J),Z(J)
  711 FORMAT(1X,I4,10X,F10.3,7X,F10.3,8X,F10.3)
  800 RETURN
      END
      SUBROUTINE MANIP(A,ACONST,MA,AMIN,AMAX,NOBS)
C
C     GENERAL ROUTINE FOR MANIPULATION (+,-,*,/) OF A VECTOR
      DIMENSION A(1)
      GO TO (11,11,12,12),MA
   11 IF(MA.EQ.2)ACONST=-1.*ACONST
      DO 20 J=1,NOBS
   20 A(J)=A(J)+ACONST
      AMIN=AMIN+ACONST
      AMAX=AMAX+ACONST
```

```
      GO TO 100
   12 IF(MA.EQ.4)ACONST=1./ACONST
      DO 30 J=1,NOBS
   30 A(J)=A(J)*ACONST
      AMIN=AMIN*ACONST
      AMAX=AMAX*ACONST
  100 RETURN
      END
      SUBROUTINE HILO(A,NOBS,AMIN,AMAX)
C
C     GENERAL ROUTINE FOR FINDING MINIMUM AND MAXIMUM OF A VECTOR
      DIMENSION A(1)
C
C     INITIALIZE MIN AND MAX VALUES
      AMIN=A(1)
      AMAX=A(1)
C
C     LOOP FOR EXTREMES
      DO 10 J=2,NOBS
      AJ = A(J)
      IF(AJ.LT.AMIN)AMIN=AJ
   10 IF(AJ.GT.AMAX)AMAX=AJ
      RETURN
      END
```